Tampere University

Ishira Uthkarshini Dewundara Liyanage

REWARD LEARNING FROM DEMONSTRATIONS FOR AUTONOMOUS EARTHMOVING

Master's thesis

Master of Science Thesis
Automation Engineering
Examiners: Prof. Reza Ghabcheloo
Dr. Nataliya Strokina
October 2020

ABSTRACT

Ishira Uthkarshini Dewundara Liyanage: Reward learning from demonstrations for autonomous earthmoving
Master of Science Thesis
Tampere University
Automation Engineering
October 2020

With the increasing complexity of specific tasks, automation engineers look at various machine learning methods as opposed to methods that require laborious task specifications. Imitation learning methods have had varying degrees of success in the past. The main drawback of imitation learning methods is their inability to adapt to newer and ever-changing problems which hinder their flexibility. Reinforcement learning aims to solve the problem by learning instead based on a rewarding mechanism. However, a reward function needs to be determined prior to carrying out reinforcement learning. A range of methods have been used to define the reward functions, which are collectively referred to as inverse reinforcement learning methods.

The objective of this research is to find a reward function for the autonomous earthmoving of a GIM Machine [1]. In this study, different inverse reinforcement learning implementations were explored. Unsupervised perceptual rewards was selected considering that it is a sample efficient method that is easy to implement on a machine without good simulations for the environment and its interactions. Based on this method, the task is broken down into stages. Demonstration data and stage labels are used to train a stage classifier. When an observation is made, it is classified into one of the stages, and the reward is calculated as a function of the distance to the next stage.

Unsupervised perceptual rewards is first used to obtain the reward function for the OpenAI Gym mountain car problem. Then q-learning is used to confirm that reinforcement learning can be applied effectively using the reward function obtained using this unsupervised perceptual rewards method. The method is then applied to demonstrations of the GIM Machine. Both low-level sensor data, as well as image features, have been used to calculate the reward. This research confirms the feasibility of using unsupervised perceptual rewards for reward function calculation and tests its robustness to changes in weather and lighting.

Keywords: Inverse Reinforcement Learning, Reinforcement Learning, Automation, Reward function, Unsupervised Perceptual Rewards

[1] A small municipal wheel loader. The body is based on Avant 635, which was converted to be driven by wire during GIM centre of excellence project (Generic Intelligence Machine 2008-2013)

PREFACE

In 2018 I was offered a scholarship to Tampere University. Needless to say I felt and still feel extremely grateful for the opportunities that presented themselves to me as a result. I have met many people in Finland who have helped me along in my Master's journey.

Having read about some of the impressive feats accomplished by robots using Machine Learning I wanted to do my research in the field. I approached Professor Reza Ghabcheloo who took me under his wing as part of the MIDAS project. He also introduced me to Dr. Nataliya Strokina, who has helped me immensely with my research work without whom I would be for a lack of better word, lost. I would like to thank both Professor Reza Ghabcheloo and Dr. Nataliya Strokina who have assisted me greatly since the start: from helping me find a good research topic to finding good research work to learn from.

I would also like to thank Wenyan Yang for helping me with the 3DCNN and Siamese Networks. Without his help obtaining good image features would have been a near impossible task. I have to thank Dr. Nikolay Serbenyuk for not only helping me understand the machine and the modifications done by our team but also for helping me collect demonstrations for the task, even on the coldest winter days.

The research team, those working under Professor Reza Ghabchello, gave me great feedback during the weekly meetings and helped me along my way and I am very grateful and thankful for that.

Lastly I would like to thank my family for their words of encouragement along the way, listening to my little setbacks and even helping me brainstorm ideas where possible.

My master's has been quite the journey and it wouldn't have been possible if not for all those who helped me. I am forever grateful for everything that people have done for me to get here.

Tampere, 25th October 2020

Ishira Uthkarshini Dewundara Liyanage

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

a	action
$\mathbb{D}$	dataset of demonstrations
E	expert
L	learner
N	number of demonstrations
p	probability distribution induced by learner's policy
ϕ	feature vector
π	policy
q	probability distribution induced by expert's policy
r	reward
s	state
T	finite Time Horizon
t	time
TAU	Tampere University
τ	trajectory
TUNI	Tampere Universities

1 INTRODUCTION

It is not easy to automate most complicated manipulator tasks, particularly those involving large machinery like machines used for earthmoving. Behavioural cloning (*Section 2.1.3*) methods have been successful in solving these automation tasks. Behavioural cloning, as the name suggests, is a method where the system attempts to clone expert behaviours by observing expert demonstrations of the task.

This research builds on top of previous research on autonomous earthmoving where our team tested behavioural cloning. *Yang et al* [2] in their research use demonstration data to learn controller parameters. However, behavioral cloning often fails to generalise to changing conditions, particularly changes in weather and lighting [3].

1.1 Problem Statement

Reinforcement learning (*Section 2.1.4*) is a branch of machine learning that can help overcome the generalisation problem. Reinforcement learning uses a system that learns based on a reward/punishment scheme. One of the main challenges with reinforcement learning is defining this reward function.

This paper attempts to overcome this challenge by computing a reward that can then be used in reinforcement learning for autonomous earthmoving. The study looks at previous research carried out on different manipulators and self-driving vehicles and attempts to find a method that is easy to implement for this somewhat unique real-world problem on a machine without good simulations of its interactions with the environment.

1.2 Research Questions

To find an effective solution to this problem, we started with five main questions that we hoped could be answered by the end of the study.

- **What sensor data is appropriate for this task?** What is the sensor data already available? Would it be possible to obtain a reliable reward function without the use of visual sensors such as cameras? If visual data is used, how should useful image features be obtained?
- **What properties should the reward function have?** Will the reward function vary linearly with respect to sensor readings. What information is available about system

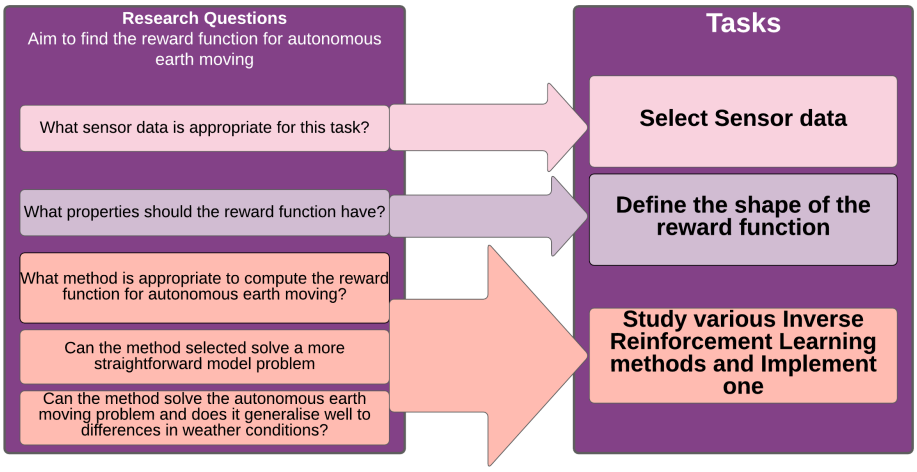

Figure 1.1. *Mapping research questions to tasks*

dynamics?

- **What method is appropriate to compute the reward function for autonomous earthmoving?** How successful have previous research been at finding a practical reward function? Can one of these methods be used to find the reward function for autonomous earthmoving?
- **Can the selected method solve a more straightforward model problem?** How successful is the method for solving a simple problem? Can it be remodelled to solve autonomous earthmoving?
- **Can the method solve the autonomous earthmoving problem and does it generalise well to differences in weather conditions?** How successful is the method at solving a real-world task? How does the robustness of the method vary with changes in weather and lighting

1.3 Tasks

The problem was broken down into tasks based on the research questions above (refer *Figure 1.1*).

- **Select sensor data** The aim is to find the most useful data for the autonomous earth moving task
  - Find descriptive features to extract
  - Visual feature extraction
- **Determine the shape of the reward function** The objective is to find a reward function that increases as we progress in the sequence. Should it increase linearly/non-linearly?
  - Understand the problem

  - Data collection and preprocessing
  - Knowledge of system dynamics (are transition probabilities available)
- **Study various Inverse Reinforcement Learning methods and implement one** The goal is to find the most effective IRL method to solve the autonomous earth moving problem
  - Study IRL methods
  - Make Comparison: ease of implementation, available knowledge of system dynamics, demonstration samples efficiency
  - Implement "best" (based on the comparison above) method on a simple continuous control problem and apply q-learning
  - Implement the method for autonomous earthmoving and obtain a reward function for demonstration data (reinforcement learning is not performed on the GIM Machine).

1.4 The Machine

The machine used for this research is a GIM (Generic Intelligent Machine)[1] based on the robotic wheel loader Avant 635 with custom power transmission and controllers designed at Tampere University.

1.4.1 Sensors Available and Where

The Machine is equipped with several sensors, as shown in *Figure 1.2*. A GNSS(Global Navigation Satellite System) and wheel odometry determine the machine's position and velocity, while IMUs determine the boom and bucket angles. The ZED stereo camera captures image data. Data from the transmission pressure sensors can determine the work done by the wheels, while data from the telescopic pressure sensors can determine the work done by the telescope. All the data is measured at 15 *fps* and sent to the controllers for processing.

1.4.2 Controller

As a somewhat complex system, the GIM Machine has several levels of control and communication. Industrial micro-controllers carry out low-level control such as CAN bus and I/O control. These low-level controllers do power management and safety functions. PC control is carried out on a PC running Realtime Simulink. The Realtime Simulink models use sensor data from low-level sensors and controllers to do tasks such as localisation. The on-board computer is a Jetson AGX Xavier. Sub-systems running on the Jetson communicate low-level sensor data and control commands to and from Realtime

[1] A small municipal wheel loader converted to be driven by wire during GIM centre of excellence project (Generic Intelligence Machine 2008-2013)

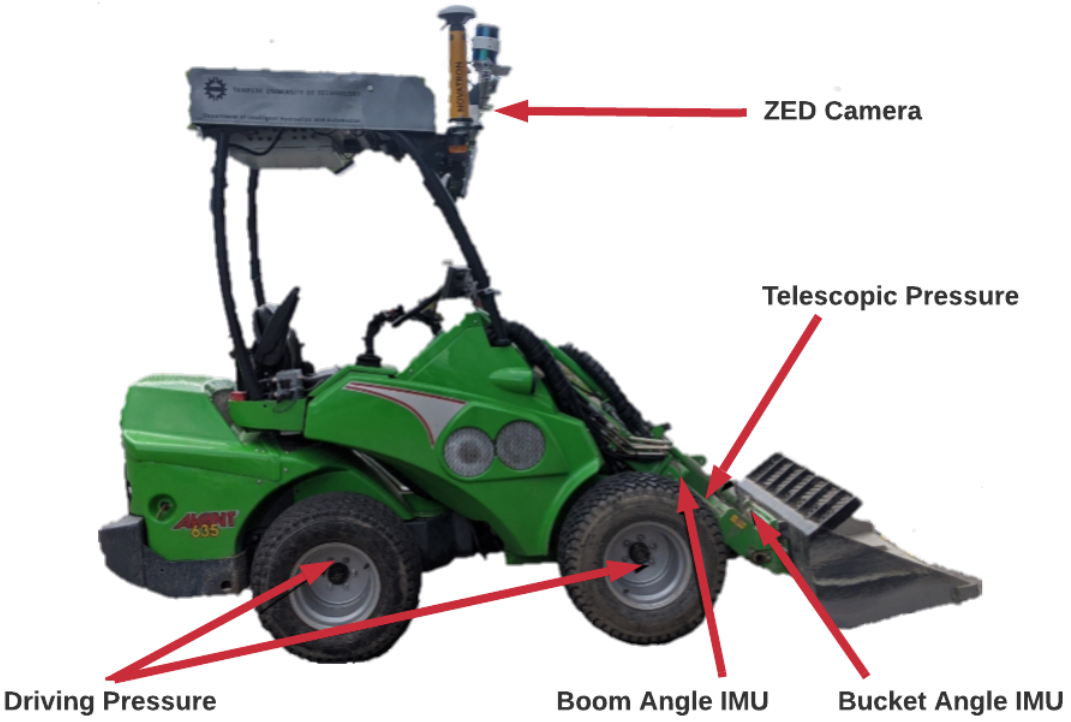

Figure 1.2. *Avant with available sensors)*

Simulink via UDP. Apart from maintaining communication with Realtime Simulink, the Jetson also carries out data collection, machine learning algorithms and closed-loop control algorithms.

1.5 Structure of the thesis

The rest of the thesis follows the following structure:

- Chapter 2: Literature Review: A study of various Inverse Reinforcement Learning techniques and reasons for selecting Unsupervised Perceptual Rewards (refer *Section 2.4.4* and *Section 2.5*) as the method of choice
- Chapter 3: Methodology : The method of implementation first on the Atari OpenAI mountain car problem in *Section 3.1*, followed by the implementation on the GIM Machine demonstration data in *Sections 3.2.1 onwards*
- Chapter 4: Results and Analysis: Similar to the methodology, results for the mountain car problem (*Section 4.1*) are followed by results on the GIM Machine (*Section* 4.2)
- Chapter 5: Conclusions: Looks at how this thesis answers the research questions discussed in *Section 1.2*

2 LITERATURE REVIEW

Robots are machines that are controlled by complex control systems. A large amount of information regarding the dynamics and processes are required to design these systems effectively. Many robots are attempting to learn these system processes and even system dynamics [4]. Today, robot learning has been used to solve various problems ranging from solving manipulator tasks [4, 5] to autonomous driving [6, 7]. Currently both optimal control approaches as well as machine learning approaches are used to solve these problems. This work attempts to use robot learning to solve a real-world problem. It aims to find the reward function for the autonomous manoeuvring of an earthmoving vehicle, the GIM Machine.

2.1 Machine Learning Background

A few key concepts must be understood before attempting to solve the task at hand. Most of these concepts are machine learning concepts that will build the background required to study reward learning.

2.1.1 Markov Decision Process

Before delving into the various types of machine learning, it is essential to introduce the Markov decision process (MDP). The Markov decision process provides a mathematical framework for designing system processes. Machine learning methods are often described by the Markov decision process [8]. The Markov decision process which is based on the Markov chain works under the assumption that the next state s_{t+1} only depends on the current state s_t. The Markov decision process is described below

- states s: a set of possible situations
- actions a: a set of possible actions
- rewards r: the reward based on state, action pairs
- policy π: mapping of states to actions
- Transition probability T: the probability of a trajectory given a state-action pair

Most reward functions and policies can be described as functions of states, actions and transition probabilities. Therefore, states, actions, rewards, policies and transition probabilities are words that would come up repeatedly in the text to follow. Other important

terminology that will come up when studying inverse reinforcement learning methods are state-visitation frequency $D^{\pi}(s)$ (*Equation 2.1*), the number of times the state s is visited on policy π, and the feature expectation $E^{\phi_k}(\pi)$ (refer *Equation 2.2*) for feature ϕ_k.

$$D^{\pi}(s) = D^{0}(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') D^{\pi}(s') \tag{2.1}$$

$$E^{\phi_k}(\pi) = \sum_{t=0} D^{\pi}(s_t) \phi_k(s_t, \pi(s_t)) \tag{2.2}$$

Having understood the Markov Decision Process it is possible to now delve into some of the most relevant types of machine learning for this research.

2.1.2 Imitation Learning

Imitation learning is a method of robot learning where the robot's objective is to imitate expert behaviour. Controllers in robotic systems determine control actions based on their observations of states; imitation learning tries to learn these controller parameters. Firstly, the expert demonstrates the task several times, over several days with varying scenarios. Expert demonstrations can be recorded, for example, as a camera feed, together with actions (i.e. brakes, speed and throttle). The data collected depends very much on the task at hand and the sensors available. The optimal policy, π^* (mapping of states to actions), is learnt by the robotics system by trying to imitate the task as done by the expert [9, 10]. In the article "An Algorithmic Perspective on Imitation Learning", *Osa et al.* [1] suggest answering a few questions before determining that imitation learning is the correct approach.

- **Why imitation learning?** The scooping technique of the GIM Machine is quite repetitive and monotonous. However, solving the optimal control problem would require having a clear idea of the system dynamics. It is much simpler to demonstrate the task at hand such that the robotic system could attempt to imitate it.
- **Who demonstrates?** The simplest way to demonstrate the task of scooping using the GIM Machine is by using teleoperation. For the autonomous earthmoving case, several demonstrations were obtained where one of the members of our team performed the task using the GIM machine on multiple days with varying weather and lighting conditions.
- **How are the expert demonstrations recorded?** The ZED camera located near the top platform of the GIM Machine records camera data. Other sensor data, including boom and bucket angles, pressures and velocities, are also recorded and stored in a CSV file.
- **What is imitated?** The goal, in this case, is to imitate the scooping technique demonstrated by an expert by learning the controller parameters.

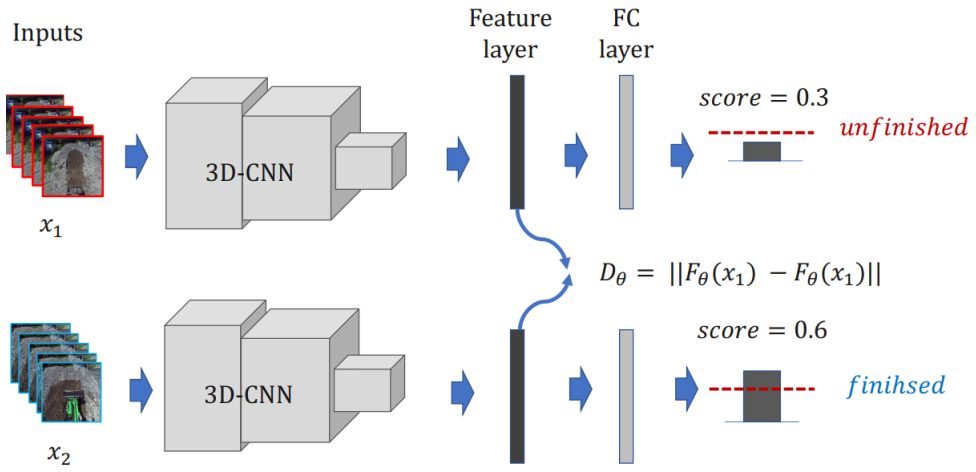

Figure 2.1. *The Siamese CNN architecture for deep visual features (Yang et al)*

2.1.3 Behavioral Cloning

Behavioural cloning, as the name suggests, is a type of imitation learning where the machine aims to clone expert behaviours (see *Algorithm 1*). An expert provides a set of demonstrations D. The robot aims to imitate the demonstrations and learn a policy π. In most robotic applications this would mean learning the controller parameters of the robot or machine.

Algorithm 1: Behavioral Cloning (*Osa et al* [1])

Result: Optimized policy parameters θ

Collect a set of trajectories demonstrated by the expert D;

Select a policy representation π_θ;

Select an object function L;

Optimize L w.r.t. the policy parameter θ using D;

The behavioural cloning method has been tested on the earthmoving problem by *Yang et al.* [2]. In the study, demonstrations of the task were carried out, and low-level sensor data such as positions, velocities, boom angle, bucket angle and transmission pressures as well as visual data was recorded. As correctly discussed by *Yang et al.* most previous research attempted to solve the automation of pile loading blind. *Yang et al.* attempts to show the advantage of using visual sensory data.

To extract useful visual features, *Yang et al.* [2] use a deep convolutional neural network of Siamese architecture and a fusion of cross-entropy and contrastive loss between successful and unsuccessful demonstrations. As shown in *Figure 2.1*, the input to the 3D convolutional neural network is five consecutive frames combined into a 3D visual tensor. The network was trained to determine if the scooping task was successful. The Siamese network was trained with successful, x_1, and unsuccessful, x_2, data pairs. The second to last layer produced a visual feature layer that was used as the features for control. During training, the objective was to maximise the Euclidean distance, D_θ, between the

extracted features for successful and unsuccessful demonstration.

$$D_\theta = ||F_\theta(x_1) - F_\theta(x_2)||_2 \tag{2.3}$$

where F_θ is the feature extraction network with parameters θ.

The contrastive loss can then be defined as shown in *Equation 2.4*.

$$L_c = D_\theta(x_1, x_2)^2 \tag{2.4}$$

This leaves the final loss used to train the network as *Equation 2.5*

$$L = \lambda_1 L_{ce}(G_\theta(F_\theta(x))) + \lambda_2 L_c \tag{2.5}$$

where L_{ce} is the standard cross-entropy loss for task classification and $\lambda_1 = 0.6$ and $\lambda_2 = 0.4$ are the weights of the losses.

After testing both neural networks and a random forest regressor to map the sensor signals to control commands, the random forest regressor produced more accurate results and was the better alternative for this task. *Yang et al.* not only find a good controller model for earthmoving but also prove that augmenting visual data to low-level sensor data increased the success rate from 40 % to 80 %.

However, most of this research was carried out during summer. The 3DCNN model was trained using summer demonstrations, and most of the testing was also carried out during summer, which means that little testing was carried out to test the robustness of the feature extractor to seasonal changes in weather. It is important to test and improve the robustness of the system and *Yang et al.* do discuss using Reinforcement Learning to improve the robustness of the controller further.

2.1.4 Reinforcement Learning

Reinforcement learning methods came into being as a solution to the generalisation problem of behavioral cloning methods. In reinforcement learning, instead of mapping the states to actions directly, the robot attempts to learn the expected behaviour based on a reward/punishment scheme [11]. The robot will traverse an unknown environment many times each time accumulating rewards and punishments (negative rewards) based on actions it takes in different situations. The learning system aims to map situations or states to actions while maximising the total cumulative reward. The goal is to find the optimal policy, π^*, to maximise the cumulative reward.

Reinforcement learning is very often used to solve autonomous driving tasks, one such study is *Kendall et al.*'s "Learning to Drive in a Day" [6]. Images from a monocular camera are used to define the state space. Then the action space is defined and includes brakes, steering and throttle. Once the state and action spaces are defined the reward function is

defined as the distance travelled before expert intervention. Finally reinforcement learning is used to learn the policy. In their research *Kendall et al.* use a very straightforward reward function, distance before expert intervention, that they claim works quite successfully. However, in most cases defining this reward function is one of the main challenges to overcome. Several methods have been used to overcome this challenge; these methods are collectively known as Inverse Reinforcement Learning methods.

2.2 Inverse Reinforcement Learning

Unlike behavioral cloning methods, reinforcement learning methods are known to better adapt to new situations. However, their ability to adapt and generalise depends significantly on their reward function. Often as is the case in the earthmoving problem, it is easier to obtain demonstrations of the task than it is to find the reward function. Inverse reinforcement learning (IRL) solves this problem by using expert demonstrations to find the reward function. In a reinforcement learning algorithm, the reward function is a function of the desired behaviours.

IRL aims to find a reward function that describes the desired behaviour. The reward function is a function of some descriptive and useful features. For a given problem, it is possible to define the reward function in many ways. Each reward function will have an optimal policy. The optimal policy found in each case, however, may not be the optimal policy for the task.

Osa et al [1] suggest looking at these questions when trying to define the correct reward function. The answers to these questions determine the efficiency and reliability of the reward function designed.

- What needs to be optimised?
- How much information is available about the dynamics of the system?
- Should the reward function be defined linearly with respect to the features?

These questions will be answered in *Chapter 3*. However, there are many important properties to take into consideration when selecting an Inverse Reinforcement Learning method.

2.2.1 Properties of the Reward function

One of the first questions that come to mind when selecting an inverse reinforcement learning method is whether the method will be model-based or model-free. Given the system dynamics, it is possible to use model-based IRL methods. A model-based IRL method learns a forward model of the system and uses it to learn a policy. Model-based methods are data-efficient and do not require sampling many trajectories to estimate the trajectory distribution. However, learning system dynamics is not a trivial task, and therefore model-free IRL methods are preferred. Model-free IRL methods aim to find the

reward function using only the data from expert demonstrations. These methods try to learn a policy without learning a forward model of the system. Model-free IRL methods require more data but usually fit non-linear features better. This research hopes to solve the autonomous earthmoving task by assuming that there is no knowledge of system dynamics. When transition probabilities are unknown, then the system dynamics cannot be defined. Without knowledge of transition probabilities, it is not easy to obtain a model.

Linearity is another important property. Does the reward/cost function vary linearly with respect to the features? How important is it that the reward function varies linearly to the features? In the case of autonomous earthmoving, the shape of the reward function is irrelevant. Either a linear or non-linear reward function may be acceptable in this case.

When selecting an IRL method there are a few other preferences. Most inverse reinforcement learning algorithms run reinforcement learning in the inner loop. That is, they are attempting to improve both the reward function and policy iteratively. An iterative method requires the system to be online, and the task must be performed either on a simulator or in real life. Is it possible to calculate the reward function without running reinforcement learning in the inner loop? An offline algorithm is preferred for the earthmoving problem as there is no good simulator with the environment and the machine's interactions for the GIM Machine to run reinforcement learning.

Another preference is a sample efficient IRL method. How many demonstrations of the task are required to obtain a good reward function? As mentioned before model-based methods tend to be relatively sample efficient, but are there model-free methods that are sample efficient? Needless to say a sample efficient method is preferred over one that requires a large number of samples as obtaining a large number of representative samples on the GIM Machine can be quite tedious.

The sections to follow, take a look at various inverse reinforcement learning methods keeping the properties discussed here in mind and eventually converging to a method that has good potential for implementation.

2.3 Model-Based Inverse Reinforcement Learning

Based on the properties discussed in *Section 2.2.1* it is clear that model-free inverse reinforcement learning methods are preferred because system dynamics are unknown. However, several model-free methods are based on model-based IRL methods. Until somewhat recently, most inverse reinforcement learning methods were model-based IRL.

2.3.1 Maximum Margin Planning

Maximum margin planning is one such model-based IRL method. Often it is hard to find a solution, a reward or cost function, that is significantly better than other functions. It is possible to converge to a solution that may not be the best. Maximum margin planning [12] aims to solve this problem by attempting to maximise the distance between the

optimal policy and other policies. This constraint can be expressed as

$$C(\tau^L) < \min_j [C(\tau) - L(\tau)] \tag{2.6}$$

where $C(\tau^L)$ is the cost for a learned trajectory τ^L and $C(\tau_j)$ are costs for other trajectories τ_j and $L(\tau_j)$ is the loss function. *Ratliff et al.* use maximum margin planning to find a good heuristic for A* for route planning for outdoor mobile robots. *Ratliff et al.* explain that this method can be applied both online and offline and can actually be run without running RL in the inner loop(refer *Section 2.2.1* for more explanation on online learning).

2.3.2 Maximum Entropy Inverse Reinforcement Learning

Most entropy optimization methods are based on *Ziebart et al.*'s [13] Maximum Entropy Inverse Reinforcement Learning method. *Ziebart et al* [13] suggest fitting a distribution that maximises the entropy. Entropy here defines the the amount of information passed in the trajectory. The idea here is when the machine is at a state and needs to decide the best action and there are two or more actions with seemingly similar results, this method chooses the trajectory that carries the most information. This method aims to obtain a distribution that maximises the entropy while trying to match the feature expectations(refer *Equation 2.7*) of the expert and the learner.

$$E_L[\phi(\tau)] = E_E[\phi(\tau)] \tag{2.7}$$

One of the maximum entropy distributions that fits this constraint is

$$p(\tau) \propto \exp(R(\tau)) \tag{2.8}$$

where $p(\tau)$ is the probability of trajectory τ. The reward function $R(\tau)$ varies linearly with respect to the feature counts $\phi(\tau_i)$.

$$R(\tau) = w^T \phi(\tau_i) \tag{2.9}$$

where w are the reward weights. Feature counts are the sum of state features in a given trajectory.

$$\phi(\tau_i) = \sum_{s_j \in \tau_i} \phi(s_j) \tag{2.10}$$

It is possible to compute the probability distribution for trajectories

$$P(\tau_i|w) = \frac{1}{Z(w)} e^{w^T \phi(\tau_i)} \tag{2.11}$$

where $Z(w)$ is the partition function. The model above aims to give exponentially higher probability for policies that reap higher rewards.

It is important to consider the transition probabilities, T, for a given action. *Equation 2.11*

can be approximated as follows.

$$P(\tau|w,T) \approx \frac{e^{w^T\phi(\tau)}}{Z(w,T)} \prod_{s_{t+1},a_t,s_t \in \tau} P_T(s_{t+1}|a_t,s_t) \tag{2.12}$$

The reward weights w are calculated to maximise the distribution 2.12.

$$w^* = argmax_w L(w) = argmax_w \sum \log P(\bar{\tau}|w,T) \tag{2.13}$$

Gradient-based optimization can be used to find the optimum. The gradient can be expressed as

$$\Delta L(w) = \bar{\phi} - \sum_{\tau} P(\tau|w,T)\phi(\tau) = \bar{\phi} - \sum_{s_i} D_{s_i}\phi(s_i) \tag{2.14}$$

where D_{s_i} is the state visitation frequency at state s_i. The expert's feature count, $\bar{\phi}$, is an average obtained over m trajectories.

$$\bar{\phi} = \frac{1}{m}\sum_i \phi(\bar{\tau}_i) \tag{2.15}$$

Ziebart et al. show that maximum entropy reinforcement learning could be used to identify route preferences and make predictions for destinations based on partial trajectories.

2.3.3 Guided Cost Learning

When designing the reward function, it is often problematic to determine the most informative features. When the system dynamics are unknown, the complexity of the problem increases further. *Finn et al.* [4] solve this problem by introducing a method they call Guided Cost Learning. Although the system dynamics are unknown, the method is a model-based IRL method because *Finn et al.* show how the forward model can be learned by representing the cost function using neural networks. Using neural networks to represent the cost/reward function has the added benefit of non-linearity. It is now possible to use features from the camera feed to design a non-linear reward function when a machine's system dynamics are unknown.

Finn et al based guided cost learning around the maximum entropy principle simply swapping out the linear cost/reward for the non-linear neural network cost/reward function. The expert samples demonstrated trajectories from *Equation 2.16*

$$p(\tau) = \frac{1}{Z}\exp(-c_\theta(\tau)) \tag{2.16}$$

where $c_\theta(\tau)$ is the unknown cost function and Z is the partition function. It is important to note that reward is negative cost. Based on this distribution the probability decreases exponentially as the cost increases. The objective function (*Equation 2.17*) for guided cost learning is the negative log-likelihood of maximum entropy distribution (*Equation*

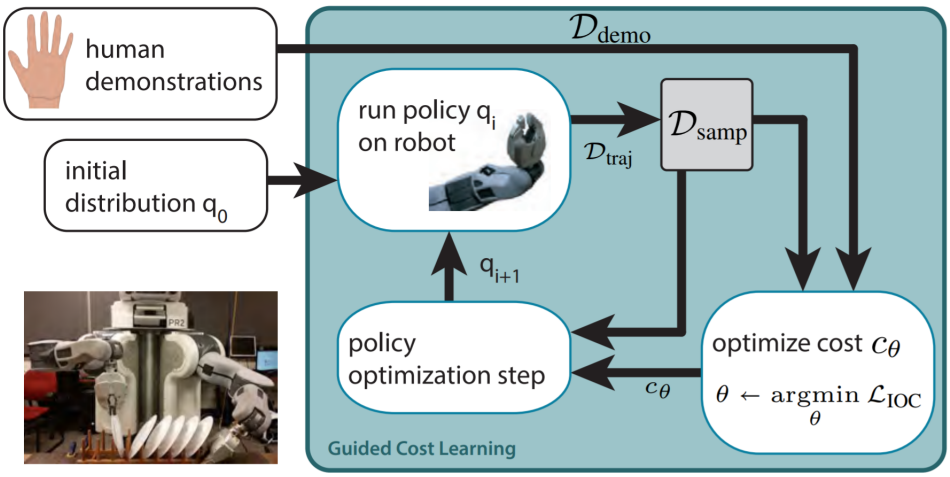

Figure 2.2. *Guided Cost Learning (Finn et al [4])*

2.16).

$$L_{GCL} = \frac{1}{N} \sum_{\tau_j \in D_{demo}} c_\theta(\tau_i) + \ln Z \tag{2.17}$$

Demonstrations are used to find the cost function $c_\theta(\tau)$. The optimal policy is computed based on the calculated cost function. This policy is then run on the machine. The machine's new trajectory is added to the demonstrations and used to calculate a new cost function. Policy optimisation is carried out again for the newer cost function to find the new optimal policy. This cycle repeats until the machine can perform the task flawlessly or until a threshold is reached (refer *Figure 2.2*).

Finn et al. use guided cost learning to show that it can be used by a PR2 robot to successfully learn a plate placement task and a pouring task.

2.4 Model-Free Inverse Reinforcement Learning

Given that the dynamics of the system are unknown, it is usually not easy to define a model as is the case for the GIM Machine. When a model cannot be defined easily, it is simpler to opt for a model-free inverse reinforcement learning method.

2.4.1 Relative Entropy Inverse Reinforcement Learning

This method [5] is built on top of the maximum entropy IRL method [13]. It also has some similarity to maximum margin planning in that *Boularias et al.* [5] suggest minimising the relative entropy between a prior trajectory $q_(\tau)$ and the learned trajectory distribution $p(\tau)$. The greatest advantage of this method over the basic maximum entropy IRL method is that it is not necessary to know the transition probabilities and therefore, it can be applied

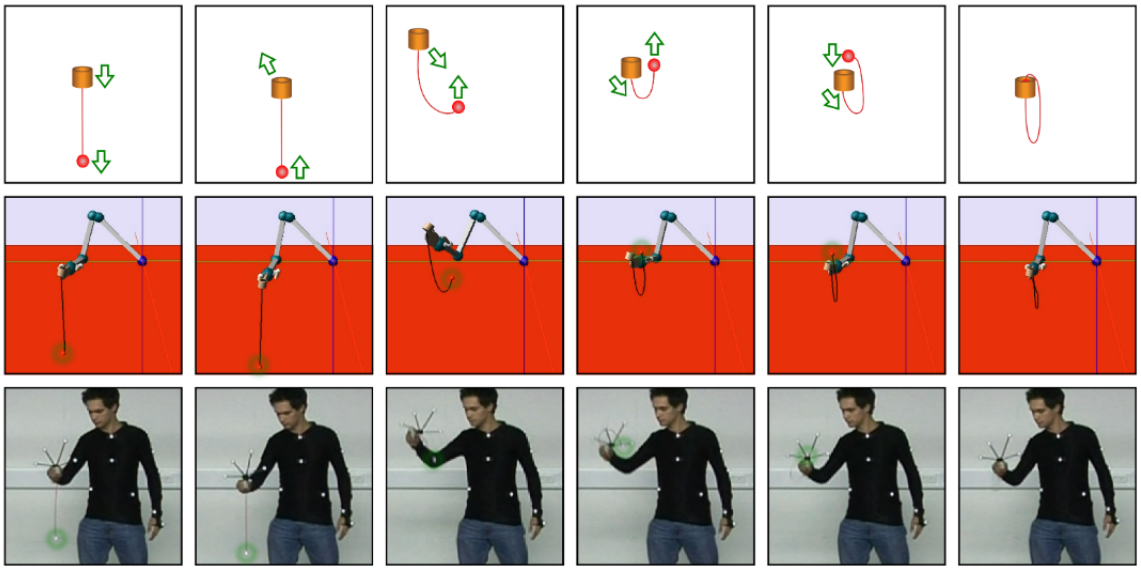

Figure 2.3. *Relative Entropy Inverse Reinforcement Learning (Boularias et al [5])*

to a system with unknown system dynamics.

$$min \sum p(\tau) ln \frac{p(\tau)}{q(\tau)} \tag{2.18}$$

subject to the difference in the feature expectations for feature $\phi_i(\tau)$ to be below a threshold ϵ_i

$$\mathbb{E}_L[\phi_i(\tau)] - \mathbb{E}_E[\phi_i(\tau)] < \epsilon_i \tag{2.19}$$

Boularias et al. show how the method was able to learn to perform the ball-in-cup task for an under-actuated robot (shown in *Figure 2.3*).

2.4.2 Generative Adversarial Imitation Learning

Similar to *Finn et al.* [4], *Ho et al.* [14] use neural networks to represent the cost function in a method called Generative Adversarial Imitation Learning. A generative adversarial network [15] consists of 2 networks working like opposing players in a min-max game. The generator network aims to generate data to fit a given distribution of data. At the same time, the adversary determines the probability that the produced data is real data from the distribution or generated data. The generator needs to produce data that is indistinguishable to the adversary. Ultimately, the data generated by the generator is so similar to the real data in the distribution that the adversary cannot determine if the data is real or generated and predicts with a probability of 0.5 that it could be real/generated data.

Ho et al. [14] in their paper use this concept to implement GAIL or Generative Adversarial Imitation Learning. This method attempts to recover a non-linear cost function given unknown system dynamics. Here the generator aims to recover the policy by generating a

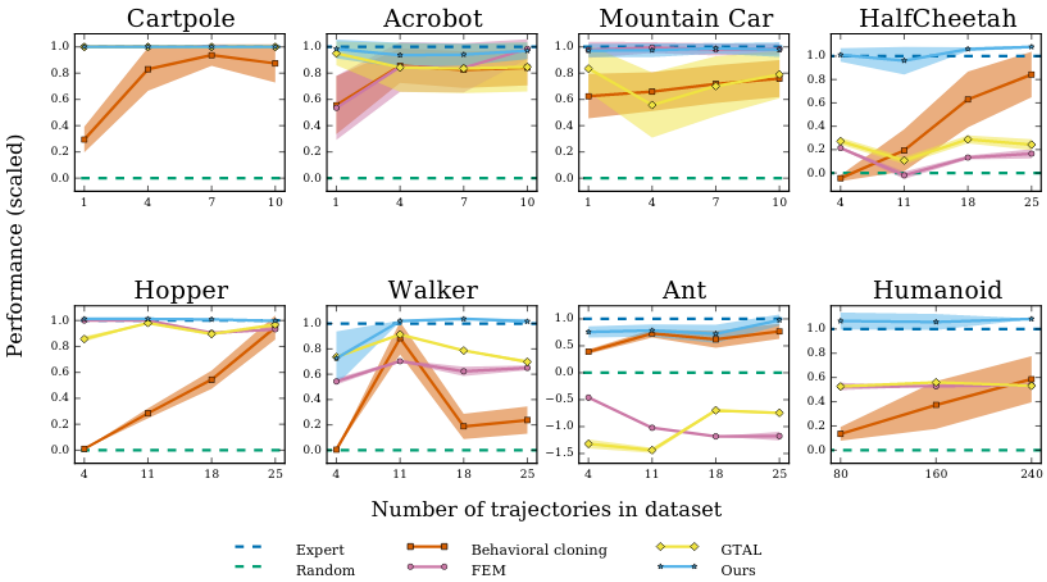

Figure 2.4. *Performance of Generative Adversarial Imitation Learning on MuJoCo control problems (Ho et al [14])*

data distribution similar to that produced by the expert. The discriminator aims to recover the reward/cost function (see *Equation 2.20*) by distinguishing expert's data from learned data.

$$c(s, a) = logD(s, a) \tag{2.20}$$

GAIL although quite sample efficient in terms of expert data, is not particularly sample efficient in terms of environmental interaction during training.

Although, *Ho et al.* don't apply GAIL to any real-world task it outperforms many other popular learning methods when attempting to solve eight MuJoCo control problems as shown in *Figure 2.4*,

2.4.3 Sub-Goal Discovery

Paul et al [16] carried out research where they use expert trajectories to identify sub-goals. In their approach, they use imitation learning to find a policy that describes expert trajectories. Then use reinforcement learning to obtain a better policy. Sub-goal discovery is another method that makes use of neural networks. Similar to the generator in generative adversarial imitation learning [14] where neural networks attempt to recover the policy, here too, neural networks are used to approximate the policy.

Figure 2.5 shows an overview of the framework used by *Paul et al* in their research. Given a state s, the sub-goal predictor predicts a sub-goal. A reward function is then defined that can be used with direct RL to obtain a policy $\pi_\theta(a|s)$.

The demonstrations are to be split to n_g sub-goals. To initialize the states are split equally among the sub-goals. The following cross-entropy loss function is optimized to obtain the

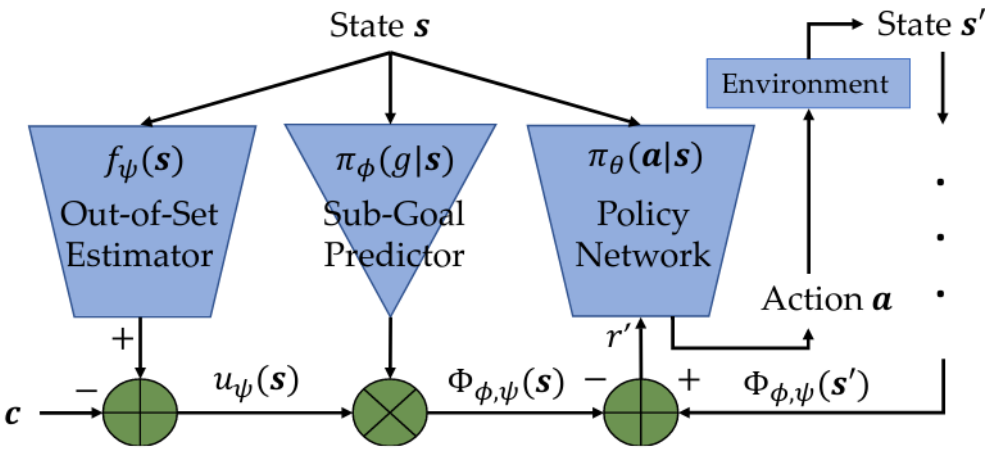

Figure 2.5. *Overview of the Sub-Goal Framework (Paul et al [16])*

sub-goal predictor $\pi_\phi(g|s)$

$$\pi_\phi^* = \arg_{\pi_\phi} \min \frac{1}{N} \sum_{i=1}^{n_d} \sum_{t=1}^{n_i} \sum_{k=1}^{n_g} -1(g_{ti} = k) \log \pi_\phi(g = j|s_{ti}) \tag{2.21}$$

The states are initialised equally among sub-goals. The reward function aims to increase the distance between the next goal state and the current goal state, thus reaching the final goal state with few steps.

$$r'(s, a, s') = \gamma * \arg\max_{j \in G} \pi_\phi(g = j|s') - \arg\max_{k \in G} \pi_\phi(g = k|s) \tag{2.22}$$

Not all states are well-represented in the expert demonstrations. The sub-goal predictor may not be able to accurately predict sub-goals for such states. The out-of-set estimator aims to adjust these predictions. The following function is optimized to learn the parameters, ψ, of the utility function, u_ψ

$$\psi^* = \arg_\psi \min \frac{1}{N} \sum_{i=1}^{n_d} \sum_{t=1}^{n_i} ||f_\psi(s_{ti} - c)||^2 + \lambda ||\psi||^2{}_2 \tag{2.23}$$

Based on the out-of-set estimator, $f_\psi(s)$, the utility function $u_\psi(s)$ calculates a utility score which indicates the likelihood that a state is represented by the demonstrations.

$$u_\psi(s) = ||f_\psi(s) - c||^2{}_2 \tag{2.24}$$

where c is a vector determined a priori. The potential function, $\Phi_{\phi,\psi}(s)$, consists of the utility score, $u_\psi(s)$, augmented to the sub-goal predictor, $\pi_\phi(g|s)$,include the

$$\Phi_{\phi,\psi}(s) = [u_\psi(s) \leq \delta] * \arg\max_{j \in G} \pi_\phi(g = j|s) \tag{2.25}$$

Given the state s and the next state s' it is possible to use the potential function to calcu-

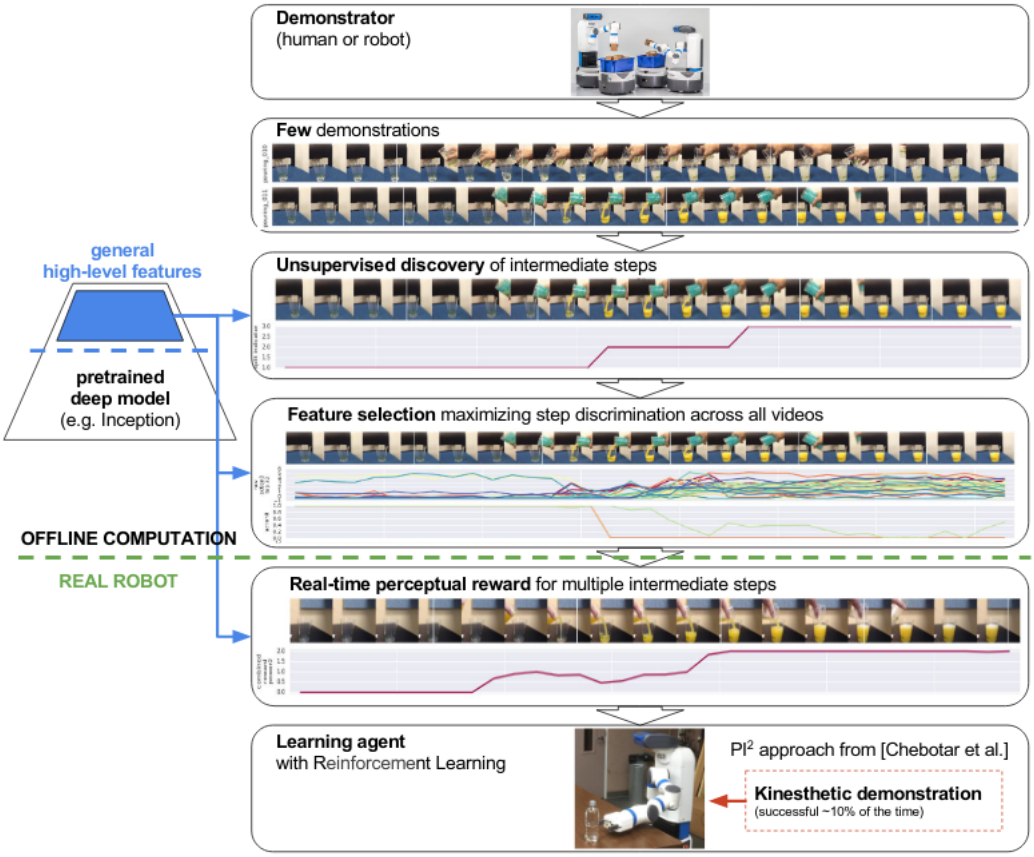

Figure 2.6. *Overview of the Unsupervised Perceptual Reward method (Sermanet et al[17])*

late reward.

$$r'(s, a, s') = \gamma\Phi_{\phi,\psi}(s') - \Phi_{\phi,\psi}(s) \tag{2.26}$$

This reward is then sent to the policy network, and reinforcement learning is carried out. This method is an online method that iteratively solves for reward and policy. The method isn't used to solve any real-world tasks but it is used to perform some simulated tasks such as ant-locomotion, where an ant learns to move towards a predefined target and ball-in-maze, where a ball tries to move to a predefined target.

2.4.4 Unsupervised Perceptual Rewards

It is interesting to note that most of the methods studied previously are online algorithms that run reinforcement learning in the inner loop. *Sermanet et al* [17] wanted to eliminate running RL to find the reward function as this meant the machine needed to be present even when designing the reward function. They prove that it is possible to design a sample efficient reward function based entirely on demonstrated data, more specifically using only features from the camera feed. Instead of using kinesthetic teaching methods like the previous methods, in this method, Sermanet et al. [17] use videos of actual human demonstrations.

Like guided cost learning [4] this method too is based on *Ziebart et al*'s maximum entropy IRL *Section 2.3.2*. Given a video demonstration, it is a tough task to find good features from raw pixels. As shown in *Figure 2.6*, *Sermanet et al* used a pre-trained deep model called Inception to find the general high-level features. Similar to sub-goal discovery, here too steps or sub-goals are defined. A recursive video segmentation algorithm will segment the video. These segments will represent the intermediate sub-goals or steps before the final goal. Step classifiers are trained to classify the steps and distinguish them. The intermediate reward functions (*Equation 2.31*) is calculated for an observation . The final reward for that observation is based on its step identification and determined by *Equation 2.32*. The idea here is that the reward increases by a factor of two as it moves to a new step further along in the task.

To simplify the problem *Sermanet et al* modify the maximum entropy equation and instead use a naïve Bayes equation that assumes that all trajectories are equally feasible, and each time step is independent of other time steps and each feature is independent of other features.

$$p(\tau) = \prod_{t=1}^{T}\prod_{i=1}^{N} p(s_{it}) = \prod_{t=1}^{T}\prod_{i=1}^{N} \frac{1}{Z_{it}} \exp(R_i(s_{it})) \tag{2.27}$$

where s_{it} corresponds to the observation for feature i at time t, T corresponds to the number of time steps and N corresponds to the number of features. As a Gaussian distribution appears to fit the above probability distribution, independent Gaussian feature distributions (*Equation 2.27*) are fitted for each step. intermediate reward functions are computed based on the top features.

$$p(s) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{[-\frac{1}{2}(\frac{s-\mu}{\sigma})^2]\right\} \tag{2.28}$$

The reward is then described as the log of the Gaussian

$$\log p(s) = -\frac{1}{2}(\frac{s-\mu}{\sigma})^2 + \log\frac{1}{\sigma\sqrt{2\pi}} \tag{2.29}$$

This can be simplified further by removing the constant

$$\log p(s) = -\frac{1}{2}(\frac{s-\mu}{\sigma})^2 \tag{2.30}$$

When considering multiple features (M number of features) this can be written as

$$R_i(s) = -\frac{1}{n}\sum_{j}^{M} \frac{(s_{ji} - \mu_{ji}^{+})^2}{(\sigma_{ji}^{+})^2} \tag{2.31}$$

where the reward $R_i(s)$ is calculated for observation s at step /sub-goal i. Steps are identified based on the similarity between time steps. It is possible to use clustering techniques or video segmentation methods to do this. The complete reward for an observation

is calculated using *Equation 2.32*.

$$R(i) = R_i * 2^{i-1} \tag{2.32}$$

Where i is the value of the current step or sub-goal.

In their paper, *Sermanet et al.* also introduce a feature selection equation, *Equation 2.33*, that scores features based on how descriptive they are. The most descriptive features have the highest score.

$$z_i = \alpha|\mu_i^+ - \mu_i^-| - (\sigma_i^+ + \sigma_i^-) \tag{2.33}$$

Where for a segment i, μ_i^+ and σ_i^+ are the mean and the standard deviation for all frames that contain the segment i and μ_i^- and σ_i^- are for those that do not contain the segment i.

Sermanet et al. successfully apply unsupervised perceptual rewards method to solve two real robotic learning tasks: pouring and door opening.

2.5 Method Selection

Based on the properties in *Section 2.2.1* comparisons were made between the different types of inverse reinforcement methods as shown in *Table 2.1*.

Table 2.1. *Properties Comparison*

IRL Method	System Dynamics	Linear	Offline possible	Sample Efficient
Maximum Margin Planning	Must be known	Linear	Yes	Yes
Maximum Entropy IRL	Must be known	Linear	No	Yes
Guided Cost Learning	Can be unknown	Non-Linear	No	Yes
Relative Entropy IRL	Can be unknown	Linear	No	Yes
Generative Adversarial Imitation Learning	Can be unknown	Non-Linear	Yes	No
Sub-goal Discovery	Can be unknown	Non-Linear	No	Yes
Unsupervised Perceptual Reward	Can be unknown	Non-Linear	Yes	Yes

Given that there is no information regarding system dynamics, it is possible to narrow the methods down to generative adversarial imitation learning, guided cost learning, sub-goal discovery and unsupervised perceptual reward. An offline algorithm is also preferred as there is no good simulator for the GIM Machine to simulate results and perform reinforcement learning in the inner loop. Both the unsupervised perceptual rewards method and

the generative adversarial imitation learning methods are capable of using only demonstrations to simulate data and calculate the reward. Being able to carry out tests offline makes implementation significantly more manageable and safer.

However, for the earth moving application, unsupervised perceptual rewards might be a better method, given that the method is more sample efficient and it is possible to obtain good results with just a handful of demonstrations. Not only that unsupervised perceptual rewards shows a method to identify the most descriptive features using a feature selection equation that gives a feature score depending on how good the feature is at describing and distinguishing a sub-task from other sub-tasks.

This research attempts to confirm the feasibility of *Sermanet et al.*'s method "Unsupervised Perceptual Rewards" for autonomous earthmoving. Unlike *Sermanet et al.* in this research, both low-level sensor data, as well as high-level image features, are used to calculate the reward function.

3 RESEARCH METHODOLOGY

Osa et al [1] (Section 2.2) suggest answering three questions when designing a reward function. In the following section (*Section 3.1*), we will attempt to answer these three questions and implement "Unsupervised Perceptual Reward" by *Sermanet et al.* [17] on the mountain car problem. Then, in the sections to follow (*Sections* 3.2.1 through 3.7), it is applied to the autonomous earth moving problem.

3.1 Continous control in simulated environment

Once unsupervised perceptual reward was selected as the method best suited for the autonomous earthmoving, it was applied to a more straightforward problem—the discrete mountain car problem from OpenAI Gym [18] (see *Figure 3.1* for the mountain car environment).

3.1.1 The Task

The first of *Osa et al.'s* questions is: What needs to be optimized? As shown in *Figure* 3.1 the car is placed on a 1D track between two mountains. The car must reach the top of the mountain (indicated by a yellow flag). However, the engine does not have enough power to reach the top of the mountain in one pass. Therefore it must move back and forth on the track to gather enough momentum to make it to the top. The task is to reach the yellow flag with the least cumulative applied energy.

3.1.2 Data available

Once the task is well defined the second question can be answered: How much information is available? The sensor data available consists of the 2D position and velocity of the car at each time step (refer *Figure 3.2* for the trajectories of position and velocity for a successful demonstration).

3.1.3 Desirable Properties of the Reward Function

The task has been well-defined, and there is good knowledge of the sensor data available, which leads to the final question: Should the reward function be defined linearly with

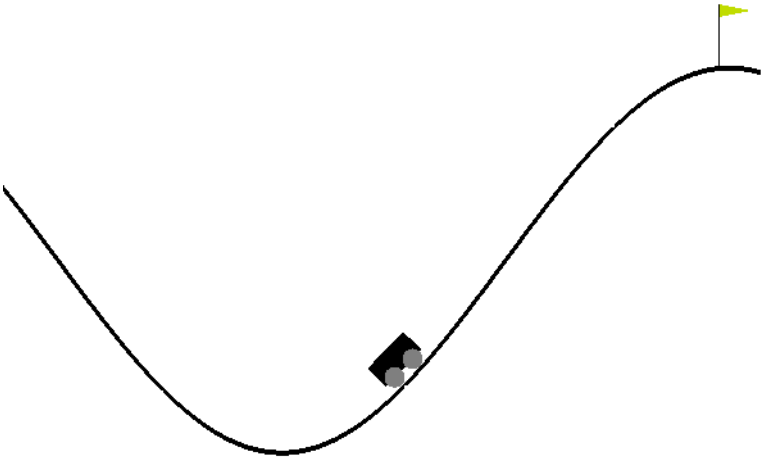

Figure 3.1. *The OpenAI Mountain Car Gym environment*

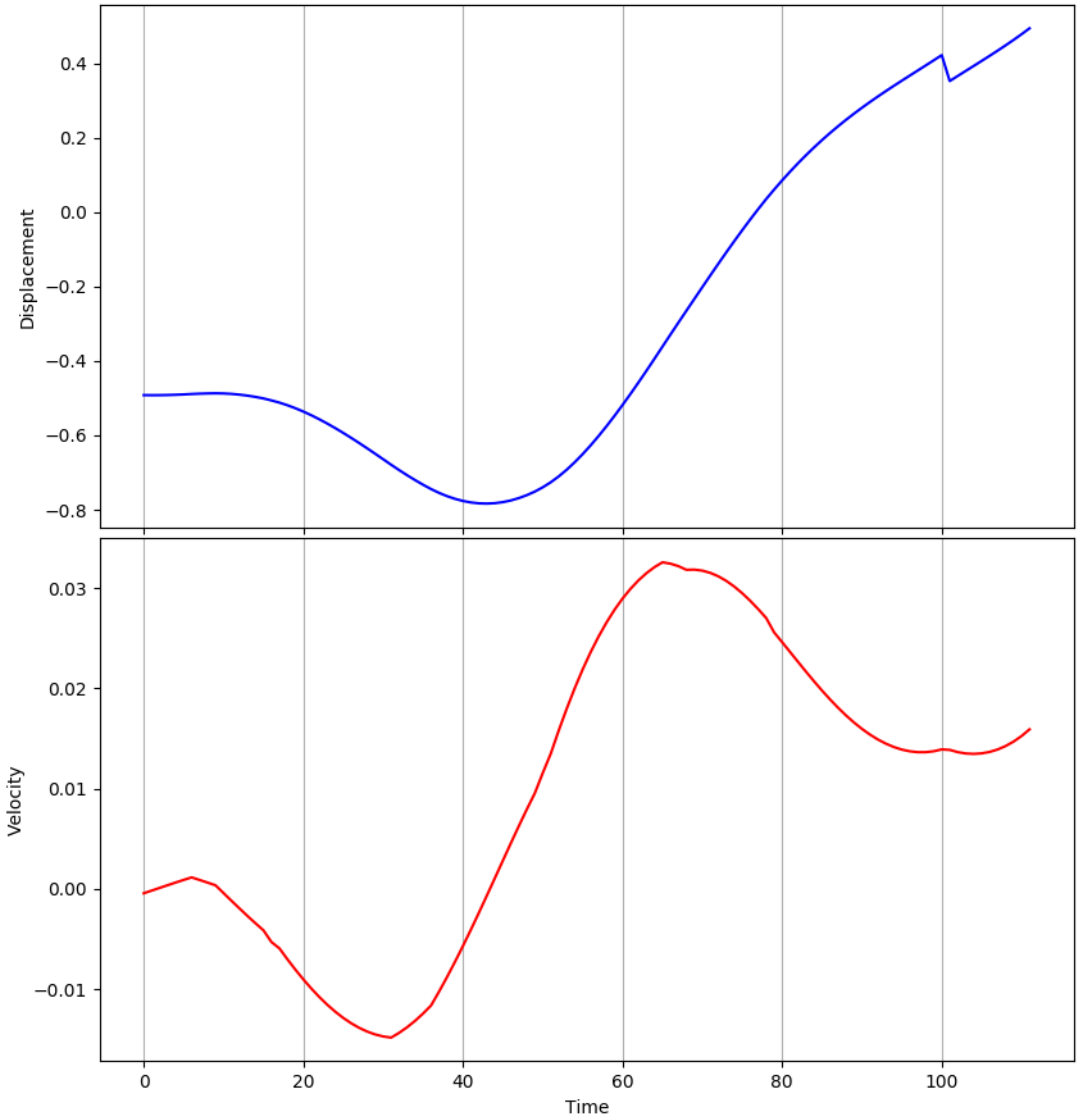

Figure 3.2. *Variation of displacement and velocity with time for a single successful demonstration of the mountain car task*

respect to the features? In order to apply q-learning to the mountain car problem, the linearity of the reward function does not matter. It is only necessary that the reward increases as the final goal is approached. As mentioned earlier, the 2D position and velocity at every time step is known. However, there is no information regarding transition probabilities. Therefore it is not possible to define a model. As a model-free algorithm, *Sermanet et al.'s* unsupervised perceptual rewards method will make a good fit.

3.1.4 Implementation

Once the task is well-defined, and a suitable method of reward calculation is determined, it is possible to move on to the implementation. The OpenAI gym provides a predefined reward function for the Atari mountain car. To find the optimal solution, q-learning reinforcement learning was performed using this predefined reward function. This optimal solution was assumed to be an "expert" demonstration. By applying q-learning using the predefined reward function, it was possible to obtain several "expert" demonstrations. After the expert demonstrations are collected, it is possible to begin testing to define a good reward function.

Testing can be broken down into two main parts:

- Reward calculation from demonstrations
- Reinforcement learning using q-learning

For reward calculation from demonstrations using unsupervised perceptual rewards, the demonstrations must be split into sub-tasks or stages. Supervised clustering was used to obtain the stages for the mountain car problem (see *Section 4.1.1* and *Figure 4.1* for why clustering was not unsupervised). All observations where the displacement was greater than 0.48 were labelled as a terminal state. Once the stages were identified, they are used to fit the stage predictor.

The intermediate reward is then calculated based on *Equation 3.1* which is a slightly modified version of *Sermanet et al.*'s version (refer *Equation 2.31*). The modified reward function was defined such that the distance from an observation s to next stage is subtracted from a maximum possible reward R_{max} (*Section 4.1.1* and *Figure 4.4* shows why the intermediate reward function was modified). The stages are defined as distributions with mean μ and standard deviation σ. If the observation was in the last stage then the distance was calculated to the terminal state. The reward for the observation was then calculated based on its stage using *Equation 2.32* as shown in *Figure 3.3*.

$$R_i(s) = R_{max} - \sum_{j}^{M} (\frac{s_j - \mu^+_{i+1,j}}{\sigma^+_{i+1,j}})^2 \tag{3.1}$$

Once good stage-classification and a good increasing reward function were obtained; q-learning (as shown in *Algorithm 2*) was applied to the model problem using the reward

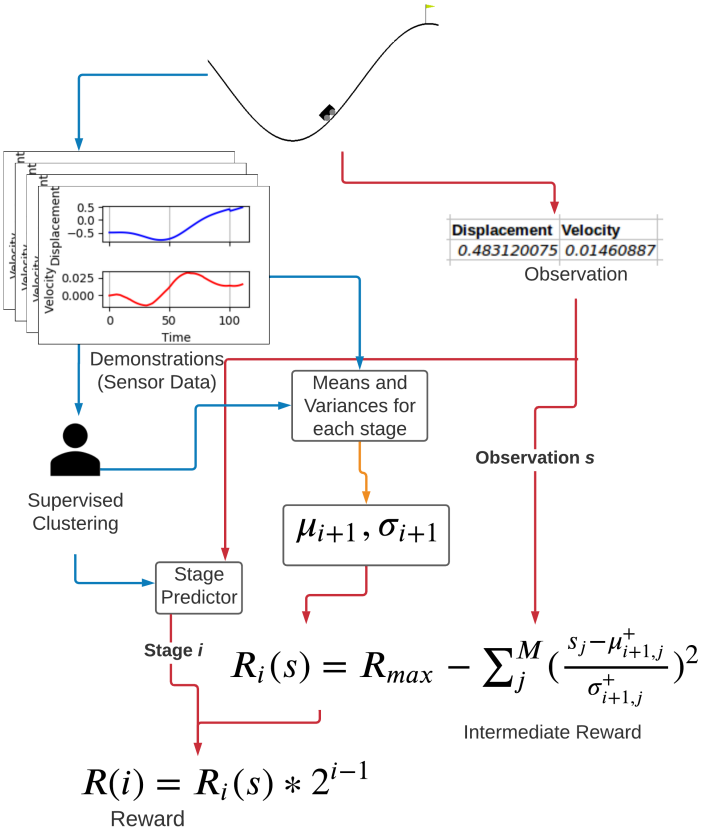

Figure 3.3. *Reward calculation from demonstration: Blue arrows represent training and red arrows represent testing*

function obtained.

Algorithm 2: Test Reward function using Q-Learning

Result: Q-Learning

for $k \leftarrow 1$ *to* $episodes$ **do**

    $done = False$;

    $t = 1$;

    while *not done* **do**

        a = getAction(s);

        s = getNewState(a);

        // Calculate Reward using Algorithm 4 ;

        R = getReward(s);

        Train Q-Learning agent ;

        if *done* **then**

            if *t<160* **then**

                Plot reward function ;

            end

            break ;

        end

        t += 1;

    end

end

3.2 GIM Machine

3.2.1 The Task

Once satisfactory results were obtained for the mountain car, and q-learning confirms the effectiveness of unsupervised perceptual rewards for reward learning, it is possible to begin implementation on the GIM Machine. The same three questions are asked here as was done for the mountain car. The first of *Osa et al.*'s [1] questions is: What needs to be optimized? The GIM Machine needs to be able to complete the scooping task regardless of changes in weather or lighting.

3.2.2 Data available

Once the task is defined, the next question must be answered: How much information is available? There are several inbuilt as well as externally attached sensors including the transmission pressure sensors, telescopic pressure sensors, IMUs measuring boom and bucket angles and a ZED camera capturing images of what is in front of the GIM Machine. All of this data needs to be collected as part of an expert demonstration.

Usually, inverse reinforcement learning begins by collecting expert demonstrations. In this application, a member of our research team drove the GIM Machine. The scooping task was carried out repeatedly over multiple days, and the data from the sensors were collected.

The data could be split broadly into two types

- winter expert demonstrations
- autumn behavioral cloning roll-outs [2]

Each demonstration consisted of images from the ZED camera at 15 fps. A CSV file (per demonstration) stores other sensor values such as transmission pressures, telescopic pressures, boom and bucket angles, velocities and boom length. The machine also recorded sensor data at 15 fps. However, there is no data available on transition probabilities. Without data on transition probabilities, we assume that system dynamics are unknown.

3.2.3 Desirable properties for Reward function

The task has been defined, and there is good knowledge about the sensor data available, which leads to the third and final question: Should the reward function be defined linearly with respect to the features? In order to perform good reinforcement learning on the GIM Machine, the main requirement is an increasing reward function. Its linearity with respect to the features does not matter.

Another important property is whether the method is model-based or model-free. Since

the system dynamics are unknown, it is not possible to define a model. Therefore, a model-free IRL method is used to solve the autonomous earthmoving task.

3.3 Implementation

The implementation can be broadly split into 2 sections

- Reward calculation
- Task completion prediction

3.4 Reward Calculation

This research applies the concept of unsupervised perceptual rewards by *Sermanet et al. [17]* for reward calculation to the more real-world problem of earth moving. Unlike *Sermanet et al.* this research not only uses image features but also low-level features such as pressures and joint angles. As seen in *Figure 3.6*, reward calculation based on this method requires some training before testing on the actual machine or demonstration data. The sensor data also needs to undergo some preprocessing. After testing various configurations of features (refer *Section 4.2.1*), the most descriptive features are identified as work done, boom angle, bucket angle and distance to the pile.

3.4.1 Preprocessing

The boom angle and bucket angle can be obtained directly from sensor data, and the distance to the pile is obtained using depth information from the ZED camera. However, the work done is calculated. Work done is calculated from telescopic pressures. The pressure is an event unlike boom angle, bucket angle and distance to the pile which are position type states (refer *Section 4.2.1* for more details). As work-done increases with time and holds the memory of previous events, it proves to be a better feature than pressure.

To calculate work-done (refer *Figure 3.4 and Equation 3.2*) the force in the boom (*Figure 3.3*) and the velocity (refer *Equation 3.4*) at the end joint must be determined.

$$Workdone = \int_0^t F_B.v_c \tag{3.2}$$

$$F_B = (a_A P_A - a_B P_B) \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{3.3}$$

where P_A and P_B are the telescopic cylinder side pressures and a_A and a_B are their

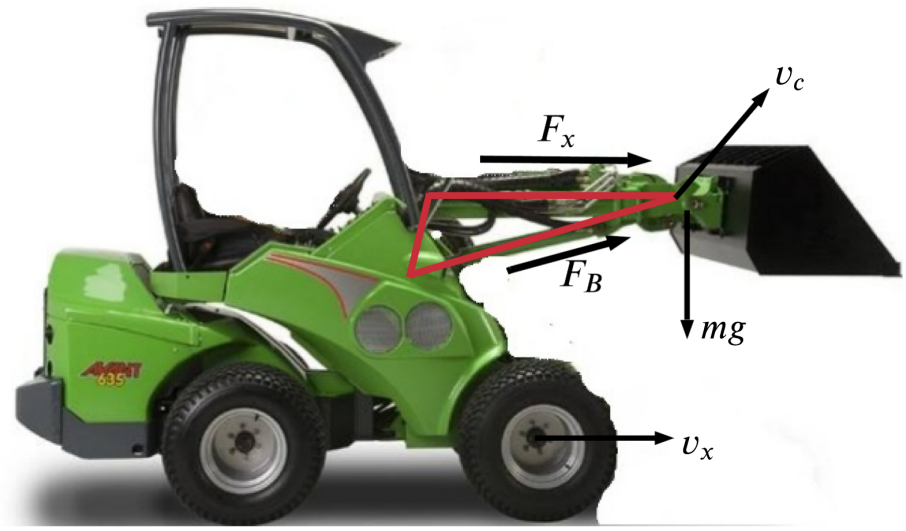

Figure 3.4. *Forces for calculating Workdone*

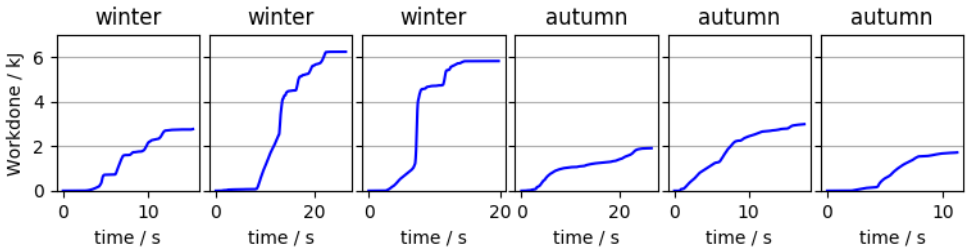

Figure 3.5. *Graphs for work done over time in winter and autumn*

respective surface areas and θ is the boom angle.

$$v_c = \begin{bmatrix} v_x - l\dot{\theta}\sin\theta \\ l\dot{\theta}\cos\theta \end{bmatrix} \tag{3.4}$$

where v_x is the forward velocity of the GIM Machine and l is the length of the boom

It is also important to note that the work done during winter is twice the work done during autumn although the shape of both graphs is similar (refer *Figure 3.5*). In order to obtain good results, work done needs to be normalized to be compatible with both winter and autumn data. Once the work done is calculated, the preprocessing is complete.

3.4.2 Training

As mentioned before in *Section 3.2.2* expert demonstrations are collected by having the expert repeat the task multiple times over multiple days, preferably with varied weather and lighting conditions. Once the sensor data has undergone the required preprocessing (refer *Section 3.4.1*) and the most descriptive features are determined, *Algorithm 3* is used to identify different stages or sub-tasks and fit a classifier.

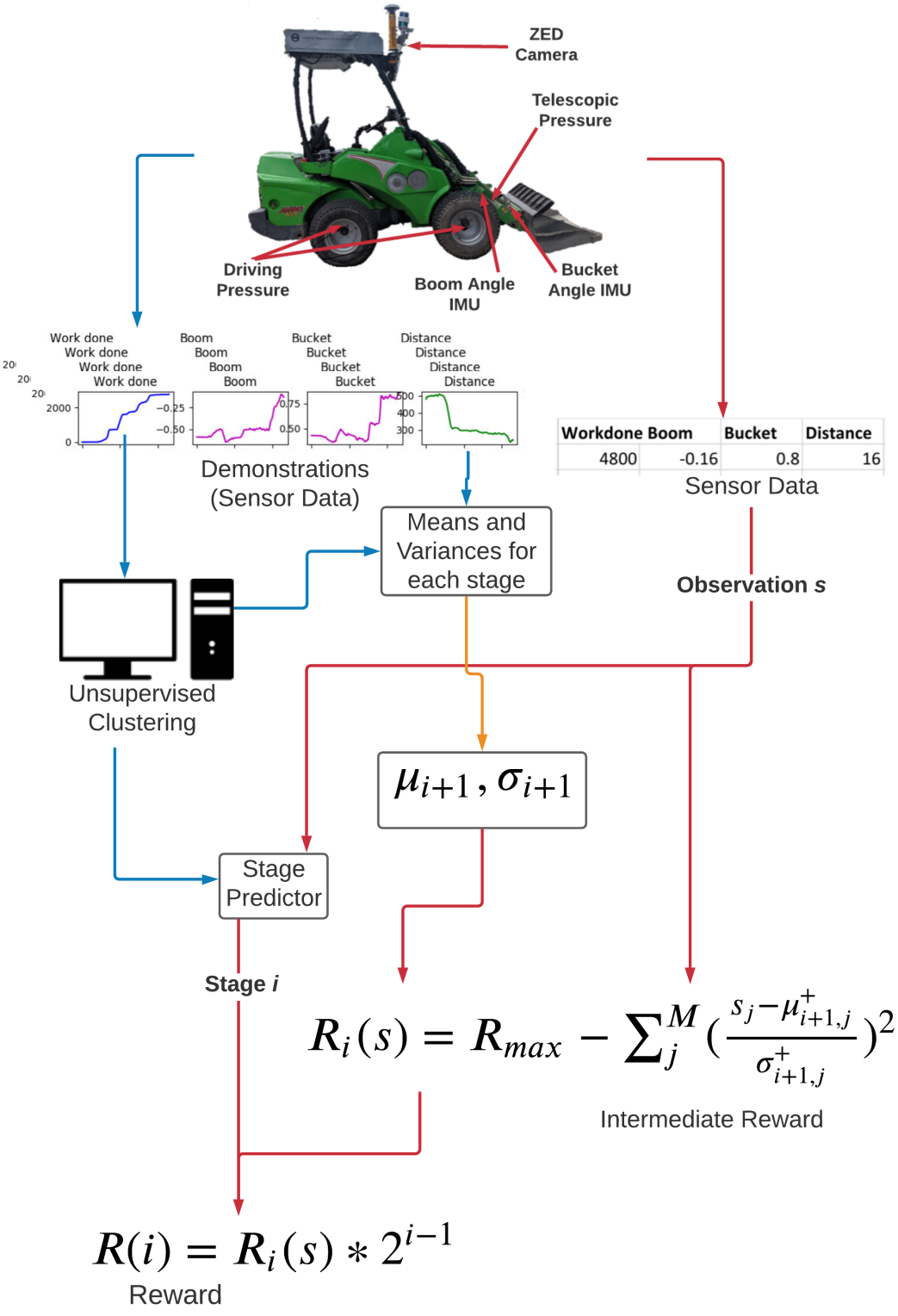

Figure 3.6. *Detailed Overview of Reward Calculation: Blue arrows represent training and red arrows represent testing*

$$\begin{array}{cccccc} \text{time} & & \text{Work} & \text{Boom} & \text{Bucket} & \text{Dist} & \text{Stage} \\ \begin{bmatrix} 0 \\ 1 \\ .. \\ .. \\ t-1 \\ t \end{bmatrix} & & \begin{bmatrix} 0 & -0.64 & 0.40 & 480 \\ 16 & -0.60 & 0.44 & 460 \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ .. & .. & .. & .. \end{bmatrix} & & & \begin{bmatrix} 0 \\ .. \\ .. \\ 1 \\ .. \\ 2 \end{bmatrix} \end{array}$$

Figure 3.7. *Data and Stages for one demonstrations after clustering*

Algorithm 3: Segment Clustering and Classifier Fitting

Result: Stage predictor, ϕ_E

Load data D: workdone,boom angle, bucket angle, distance to the pile ;

Set cluster centers ;

K-Means clustering to obtain segments;

Fit stage predictor using clustered data ;

Calculate μ and σ for each segment ;

$\phi_E = \phi_E \cup [\mu, \sigma]$;

When testing various clustering methods, it became apparent that both kmeans clustering as well agglomerative clustering gave good results. However, kmeans clustering was used in this case, as this meant it was possible to initialize cluster centres. At this point, the demonstration data will have labels. Each data record will have a stage. In this research, the demonstration data was split into three separate clusters. These data clusters represent the three stages of the GIM Machine's scooping task. The data records are now all labelled with a stage (0-2 inclusive). The labelled data will be used to fit the stage predictor. The stage predictor is a classifier used to classify the observations to stages during testing. Both the knn classifier and support vector machines presented good results during classification.

As seen in *Figure 3.6*, the means, μ, and variances, σ for each stage are calculated. These means and variances are used to represent the expert data and are required during testing. The mean, μ_i, for a single-stage, i, is a vector holding the mean, $\mu_{i,j}$, for each feature, j, in that stage. For these experiments, the vector holds the four elements for work done, boom angle, bucket angle and distance to the pile. Variances are held in a similar four-element vector. If there are three stages, then there will be a total of three sets of vectors or six vectors in total. These set of vectors and the classifier will hold all the information needed for testing.

3.4.3 Testing

When an observation is made, it will have to undergo similar preprocessing as was carried out on the demonstration data. Work done and distance to the pile will need to be

determined. Then this array of features (4 elements) (refer *Algorithm 4*) will be sent to the Stage predictor which will classify the observation to a stage (0-2 inclusive) based on the features observed(see red arrows in *Figure 3.6*).

Then based on the stage, the mean, μ_{i+1}, and variance, σ_{i+1}, of the next stage are obtained. If the observation is from the last stage (stage i=2 in this research), then the mean, μ_i, and variance, σ_i, of that stage are used. Unlike for the mountain car, a terminal state was not defined as good results were observed without it.

The intermediate reward is then calculated using the following *Equation 3.5* which is derived based on *Sermanet et al's Equation 2.31*.

$$R_i(s) = R_{max} - \sum_{j}^{M} (\frac{s_j - \mu^{+}_{i+1,j}}{\sigma^{+}_{i+1,j}})^2 \tag{3.5}$$

Once the intermediate reward is calculated in this way. It is multiplied by a factor of 2 based on the stage to obtain the final reward for that observation (*Equation 3.6*).

$$R(i) = R_i(s) * 2^{i-1} \tag{3.6}$$

Algorithm 4: Calculate Reward

Result: Reward R given demonstrations $\mathbb{D}$

$\phi_E = []$;

for $segment$*=0 to length(segments)* **do**

  // μ_i here is a list containing mean across $segment$ for each feature ;

  μ_i = mean($\mathbb{D}$[segment]);

  // σ_i here is a list containing standard deviation across $segment$ for each feature ;

  σ_i = stdev($\mathbb{D}$[segment]);

  $\phi_E = \phi_E \cup [\mu_i, \sigma_i]$;

end

if *observation* s **then**

  Use stage predictor to classify s ;

  Obtain μ and σ to next cluster center ;

  Calculate $R)i(s)$ using *Equation 3.5* ;

  $R_i(s) = R_{max} - \sum_{j}^{M} (\frac{s_j - \mu^{+}_{i+1,j}}{\sigma^{+}_{i+1,j}})^2$;

  // i is the segment number ;

  Calculate $R(i)$ using *Equation 2.32*;

  $R(i) = R_i(s) * 2^{i-1}$;

end

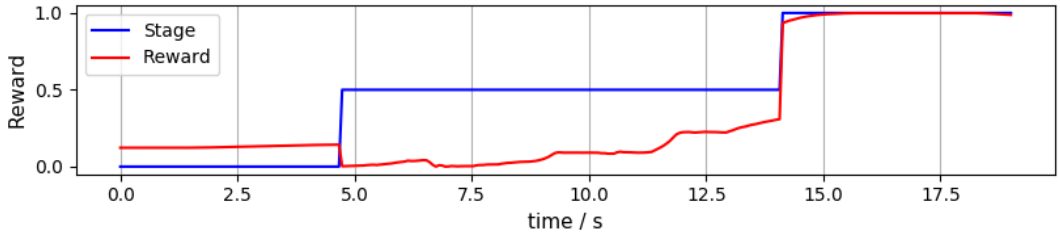

Figure 3.8. *Reward calculation unsuccessful* $R_{max} = 100$

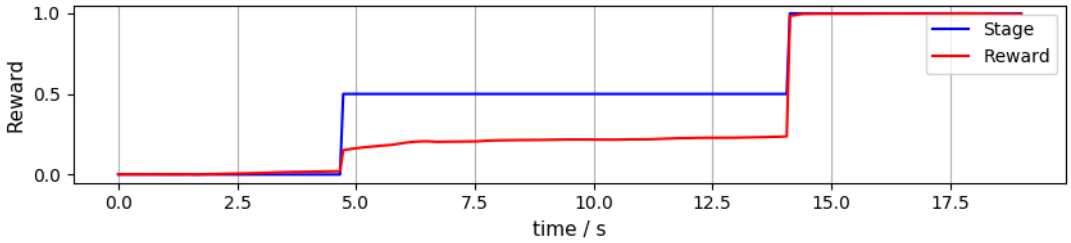

Figure 3.9. *Reward calculation successful* $R_{max} = 600$

3.4.4 Initializing R_{max}

Trial and error is used to initialize R_{max}. A negative intermediate reward $R_i(s)$ is problematic because when this reward is substituted to *Equation 2.32* this negative reward would become more negative, resulting in a dip in the middle of the reward function. This means although the task is in the second stage the reward is lower than it was in the first stage. Such a dip in reward could make reinforcement learning very difficult as the algorithm may never be able to converge to a good solution.

Initially, R_{max} is set to a relatively low value (i.e. 100-200) and the algorithm is run on a successful demonstration and the reward is calculated. The reward must increase with time. If the reward goes down as shown in *Figure 3.8* (the reward is lower at the start of the second stage than it was in the first stage) then make an increment of 100/200. Repeat this until the reward increases with time (refer *Figure 3.9*).

3.5 Task Completion Prediction

This research adds another layer to *Sermanet et al.'s* [17] method. Here a task completion predictor is added that would indicate the task is completed based on visual data.

As the expert performs the task, the GIM Machine also collects visual data. To train the task completion predictor visual features must first be extracted. For visual features extraction, the pre-trained model from *Yang et al.*'s [2] research is used. It consists of a 3DCNN that uses five images as input and outputs eight image features (refer *Section 2.1.3* to see how *Yang et al.* trained the 3DCNN). These features are then used to fit the task completion predictor as shown in *Figure 3.10*. The task completion predictor is simply a binary classifier that would output whether the task has reached completion.

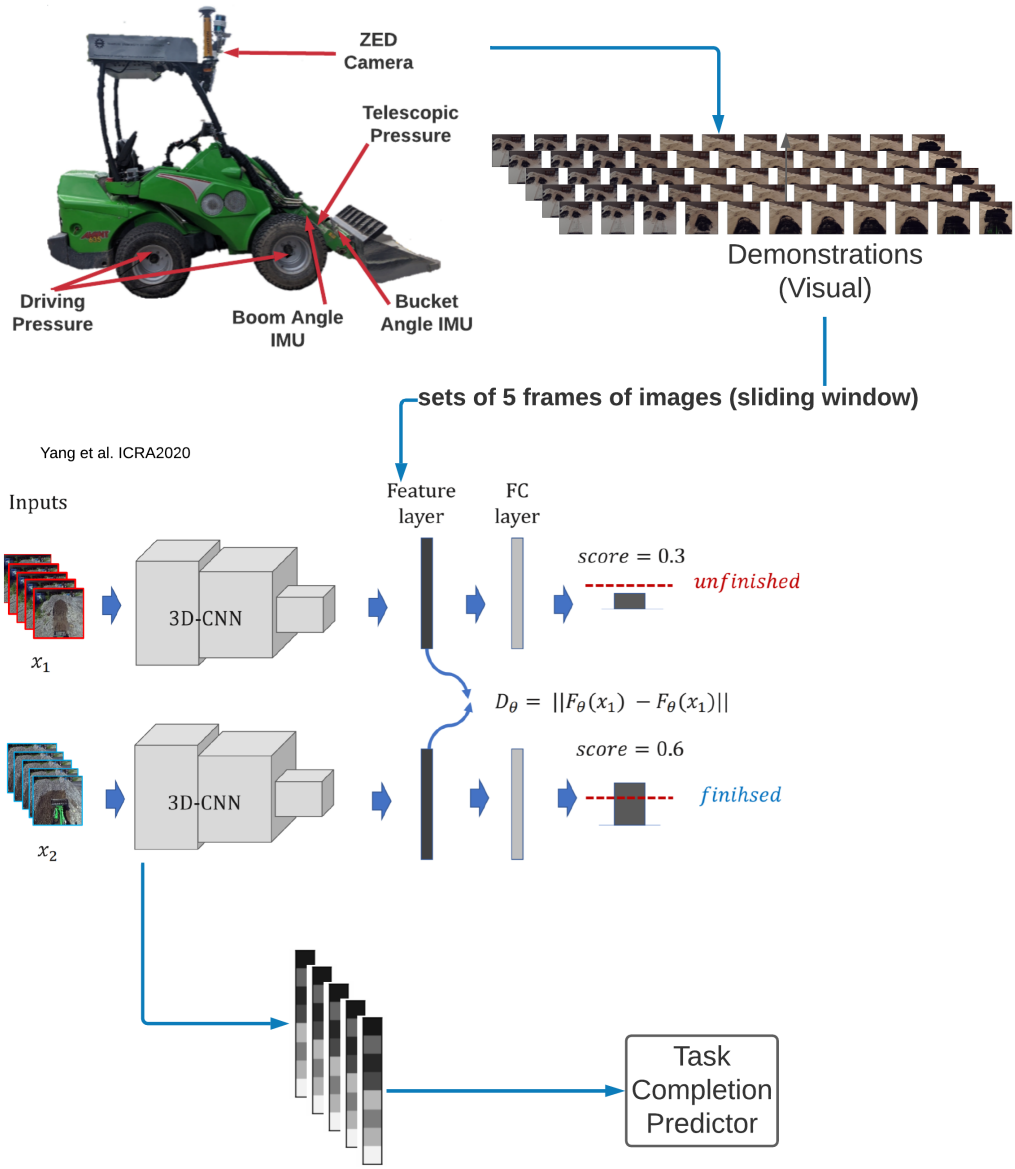

Figure 3.10. *Overview of Task Completion Predictor Training (see Figure 3.13 for Entire System)*

As done during training a sliding window is used here where five frames of the visual data is input to the 3DCNN and eight visual features are output as shown in *Figure 3.11*. These features are sent to the task completion predictor for classification.

3.6 Reward Calculation with Visual

The visual features extracted by *Yang et al*'s [2] pretrained model were used to determine if the task was successful. Could these features be used to for reward calculation? The research also looked at the feasibility of using visual features trained to determine scooping success for reward calculation. The eight visual features extracted from the 3DCNN are combined with the demonstrations(containing low-level sensor data) prior to being sent for unsupervised clustering during training (blue arrows in *Figure 3.12*). Similarly

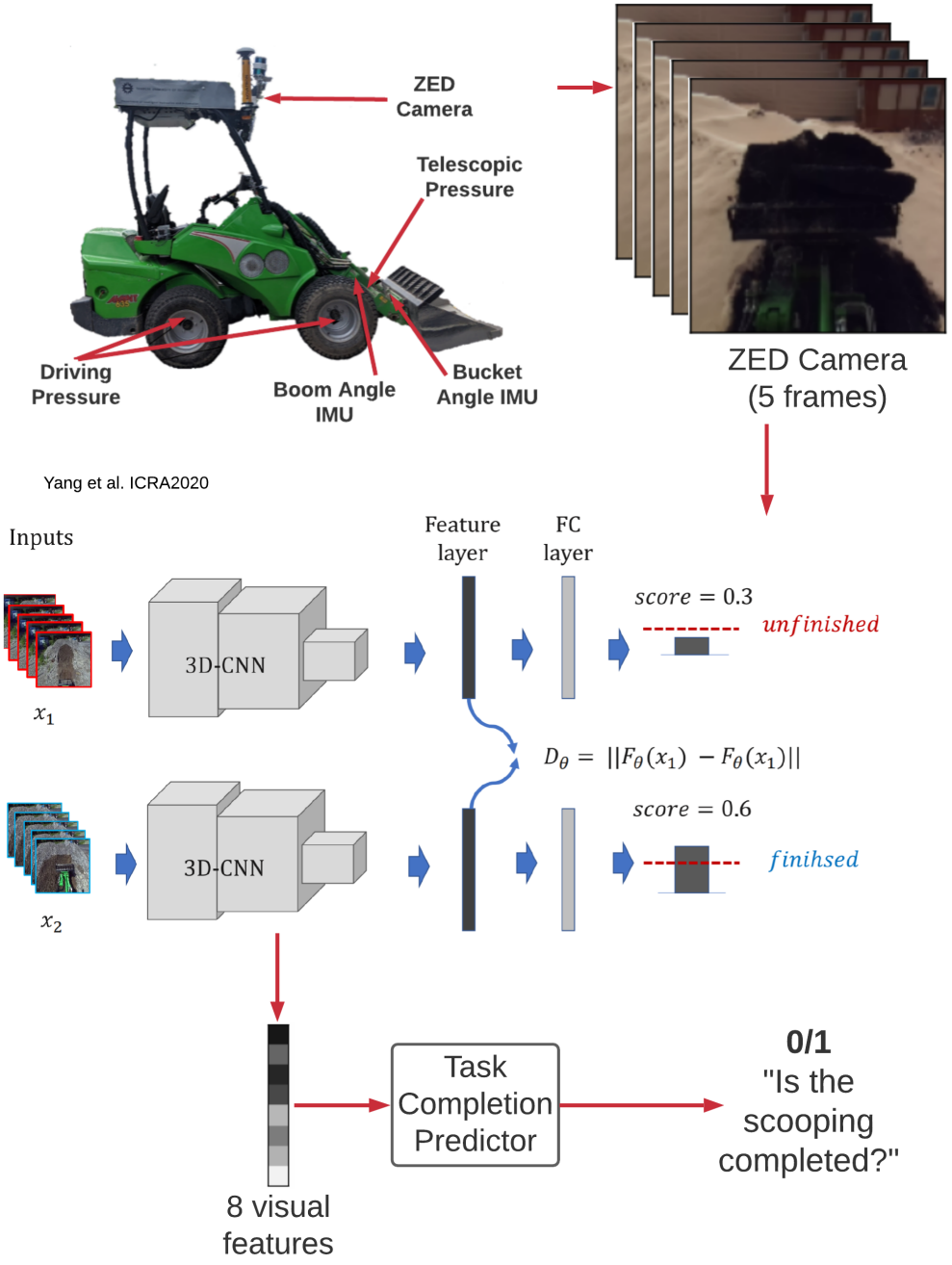

Figure 3.11. *Overview of Task Completion Predictor Testing (see Figure 3.13 for Entire System)*

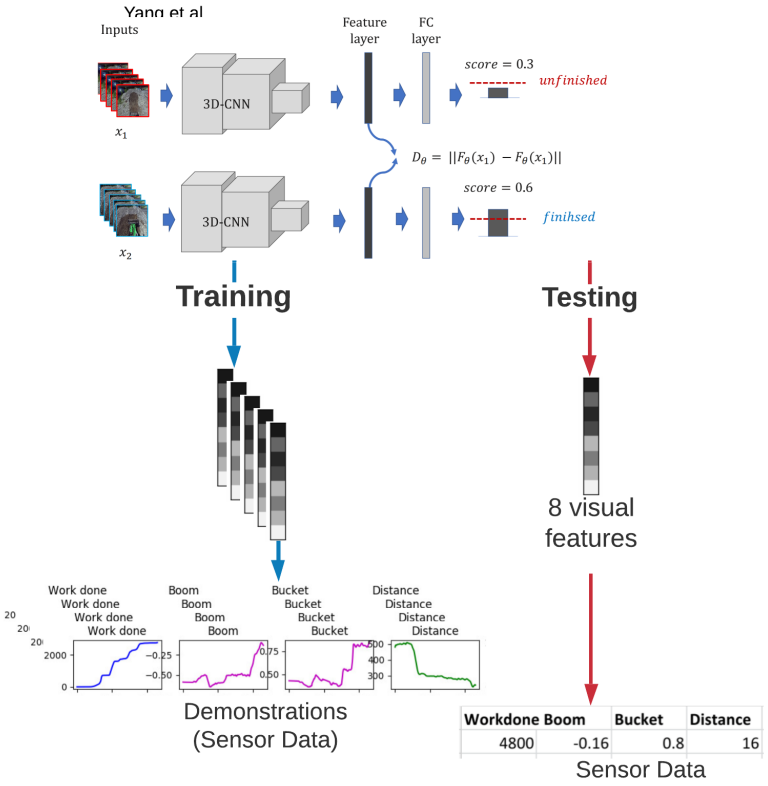

Figure 3.12. *Add Visual Features to Feature vectors (see Figure 3.13 for Entire System)*

during testing the eight visual features are combined with the observation from low-level sensors (red arrows in *Figure 3.12*). Now the vectors holding feature data would be two twelve (four low-level features + eight high-level image features) element vectors containing the means and variances.

It is important to note that R_{max} may need to be recalculated as shown in *Section 3.4.4* when new features are added to the feature vector

3.7 Testing on Multiple Demonstrations

Testing a single demonstration is usually not enough. *Algorithm 6* is used to perform testing on multiple demonstrations. The training and testing was carried out on 3 main types of demonstrations, D. (refer *Table 3.1*).

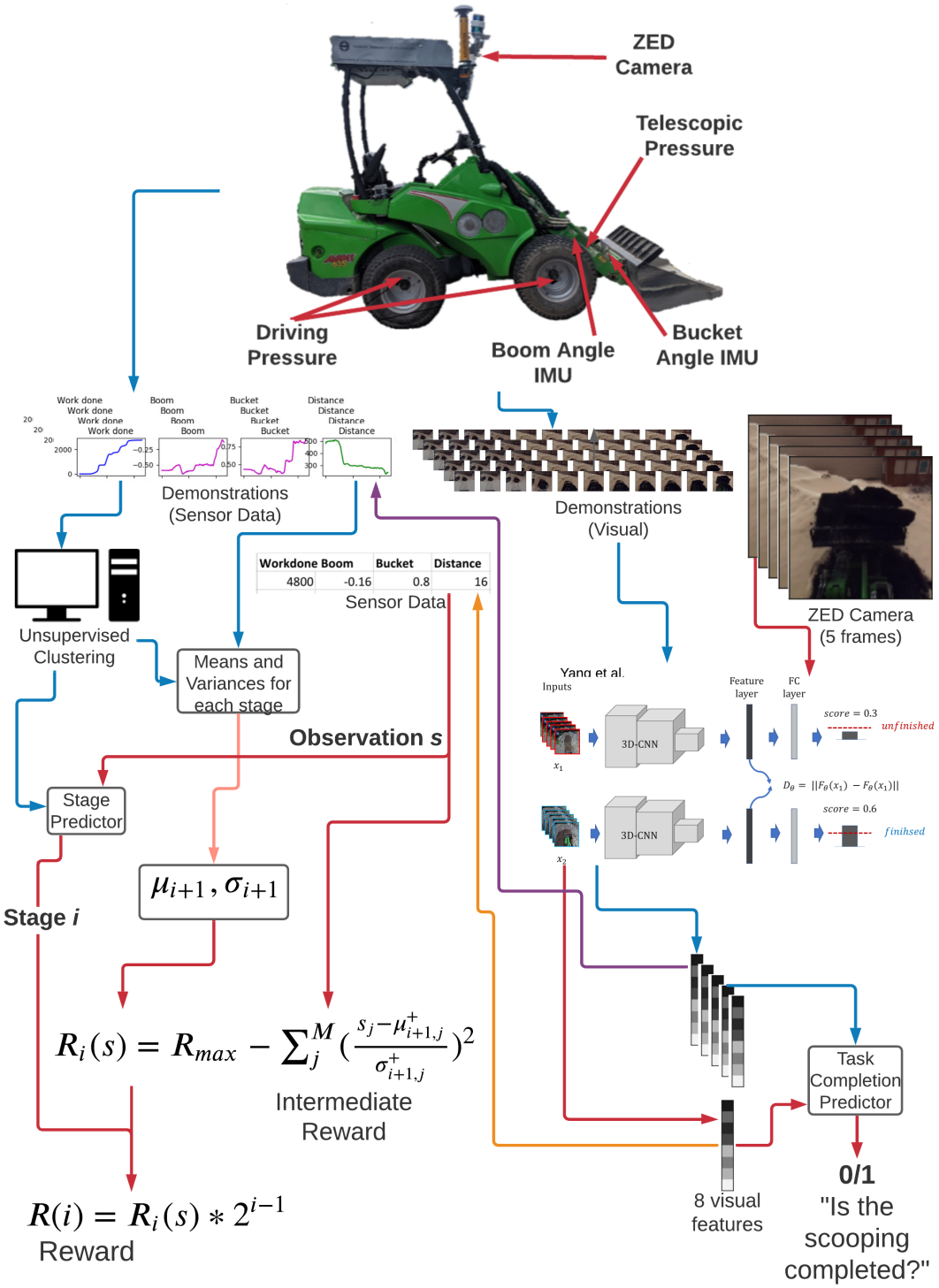

Figure 3.13. *Detailed Overview of Entire System with Visual Features for Reward Calculation: Blue arrows represent training and red arrows represent testing and Purple and Yellow represent the addition of visual features for reward calculation*

Table 3.1. *Types of Training and Testing*

Type	**Training**	**Testing**
Mixed	All demonstrations except those tested (usually 80 % of the demonstrations)	Demonstrations tested (usually 20 % of the demonstrations)
Testing winter	All autumn roll-outs (from behavioral cloning, previous research [2])	A/Some winter demonstration/s
Testing autumn	All winter demonstrations	A/Some autumn rollout/s

Algorithm 5: Test One Demo

Given ϕ_E and classifier ;

for s *in* D **do**

  Get observation s ;

  Calculate Reward for s using *Algorithm* 4 ;

end

Algorithm 6: Test Multiple Demos

Split demonstration data to Training and Testing Data (refer *Table 3.1*);

Run Algorithm 3 to obtain classifier and ϕ_E ;

Run *Algorithm* 5 for all Testing Data ;

4 RESULTS AND ANALYSIS

Similar to the methodology in this chapter too, results for the mountain car problem (*Section 4.1*) are followed by results on the GIM Machine (*Section* 4.2 onwards).

4.1 Continous control task in simulated environment

The OpenAI mountain car problem is straightforward. The goal here is to drive the mountain car up the mountain, but the engine is not strong enough to make it to the top in a single pass. Therefore the car must drive back and forth and build enough momentum to reach the top. The car is on a 1D track and the sensor data available at any given time step consists of the 2D position and velocity. Since there is already a predefined reward function, it is possible to use this reward to apply q-learning and obtain several successful demonstrations.

4.1.1 Reward Calculation for successful demonstrations

Reward calculation using unsupervised perceptual rewards is performed initially on successful demonstrations. The data was trained on a few(4) successful demonstrations and then tested on one other successful demonstration (refer *Figures 4.6*). Since the problem is simple and straightforward, it is possible to obtain a representative reward function with as few as 4 data sets. The number of demonstrations increased as good results were obtained; those results were added to the successful demonstrations as well.

To perform unsupervised perceptual rewards, the task is first broken down into sub-tasks or stages based on the demonstrations. Initially, kmeans clustering was used to identify these different stages. However, the stages produced in this way weren't satisfactory (refer *Figure 4.1*). Therefore, supervised clustering was carried out. The stages were split at the points where the acceleration of the car was 0 (refer *Figure 4.3*). When the position was greater than 0.48, the task was at a terminal state. Once the data was correctly clustered, it was possible to fit the stage classifier.

It was also possible to obtain stage classification accuracy because the stage identification was carried out by supervised clustering (see *Figure 4.3*). Several classifiers were tested for stage classification (as seen in *Table 4.1*) until good classification as well as reward calculation were observed (*Figure 4.6* shows good stage classification and instantaneous reward). Interestingly although the KNN classifier had a high stage classification

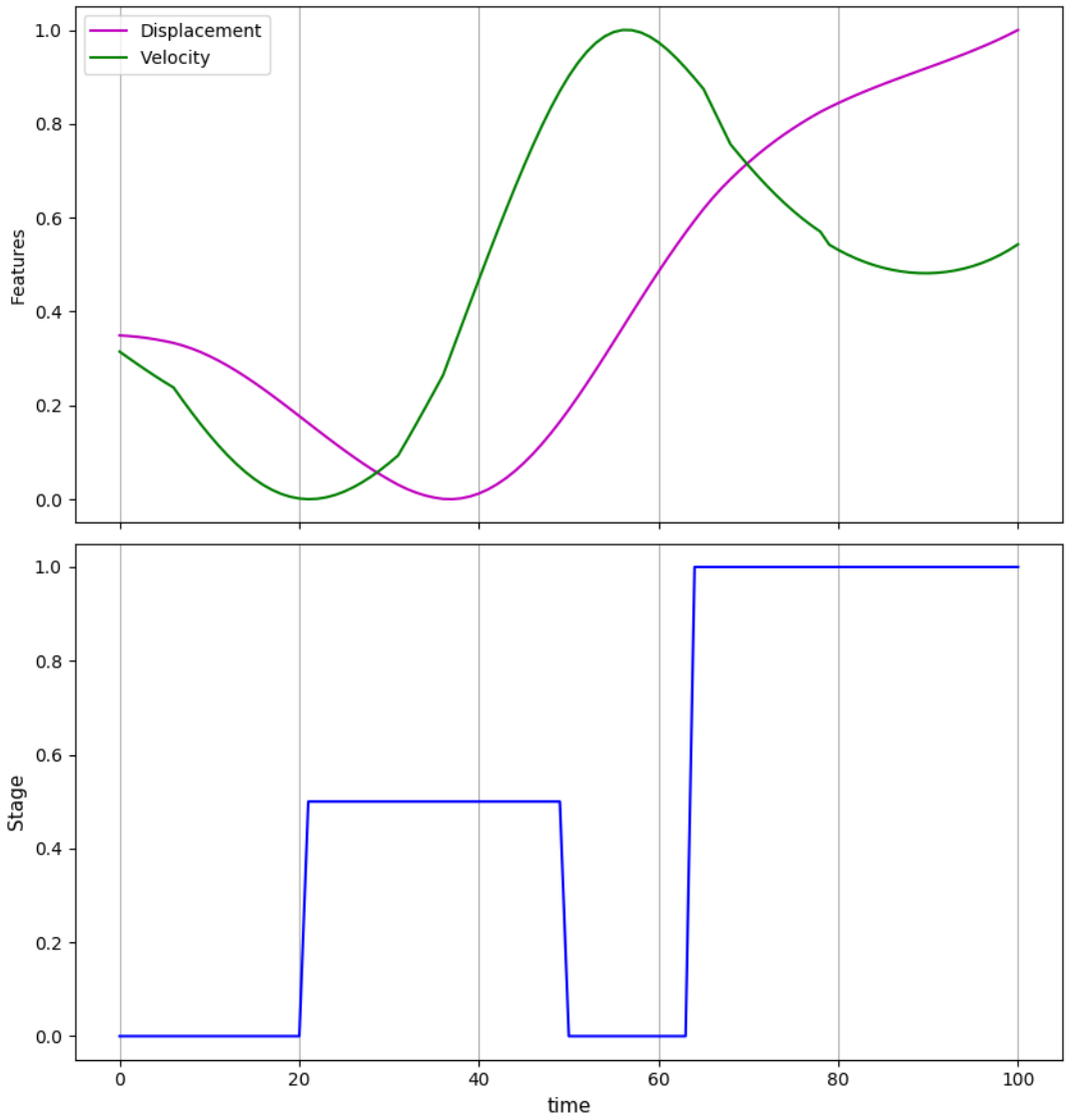

Figure 4.1. *Stages/Segments for a single successful demonstration using KMeans clustering*

accuracy (see *Table 4.1*) it repeatedly misclassified some samples in the third stage, which would result in an impaired reward function (refer *Figure 4.2* for one example of bad KNN Classification).

Following good classification (see *Figure 4.3* for good classification) the next step is to calculate the reward. When calculating the reward, initially tests were carried out using *Sermanet et al*'s original equation *Equation 2.31*. Where the distance from the observation to the current stage was used as the reward. As seen in *Figure 4.4* the reward decreases when the task is nearing completion. Drops in reward are characteristic of a bad reward function that will fail to converge when reinforcement learning is applied.

Therefore in this research instead, the distance to the next stage is calculated as shown in *Figure 3.1*. All observations where the displacement was greater than 0.48 was identified as a terminal state. So when an observation was classified to the last stage, then the μ and σ of the terminal state are used as μ_{i+1} and σ_{i+1}. Once good results were obtained

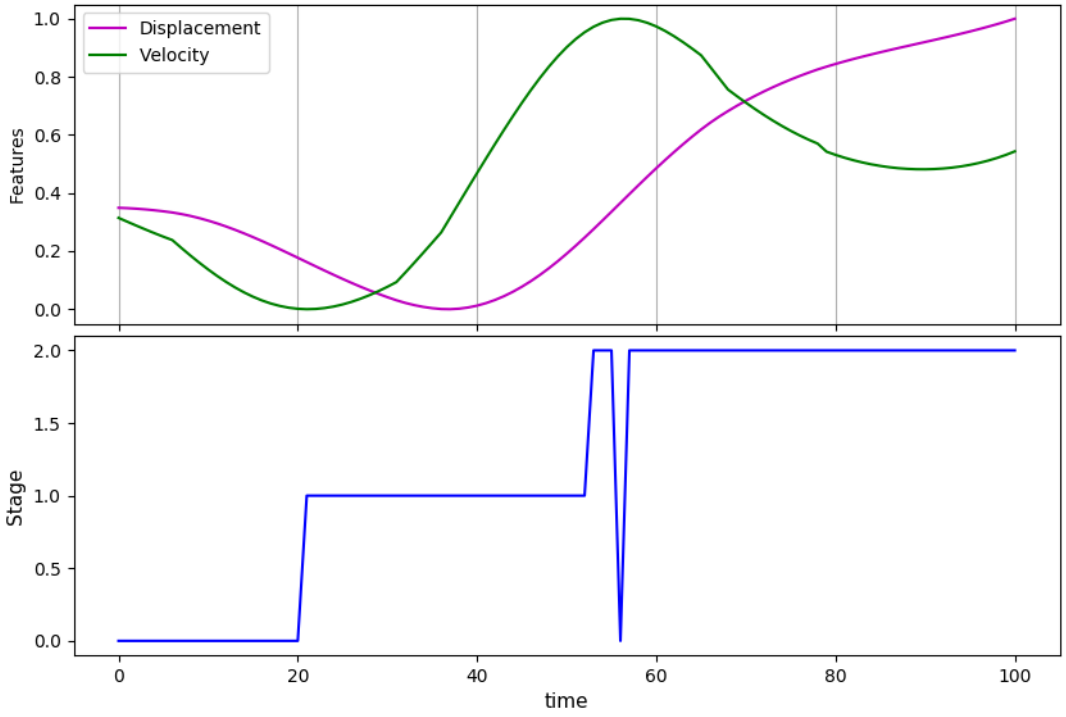

Figure 4.2. *Bad Reward Function for the mountain car problem (KNN Classifier and* $R_{max} = 16000$*), bad classification at timestep = 52*

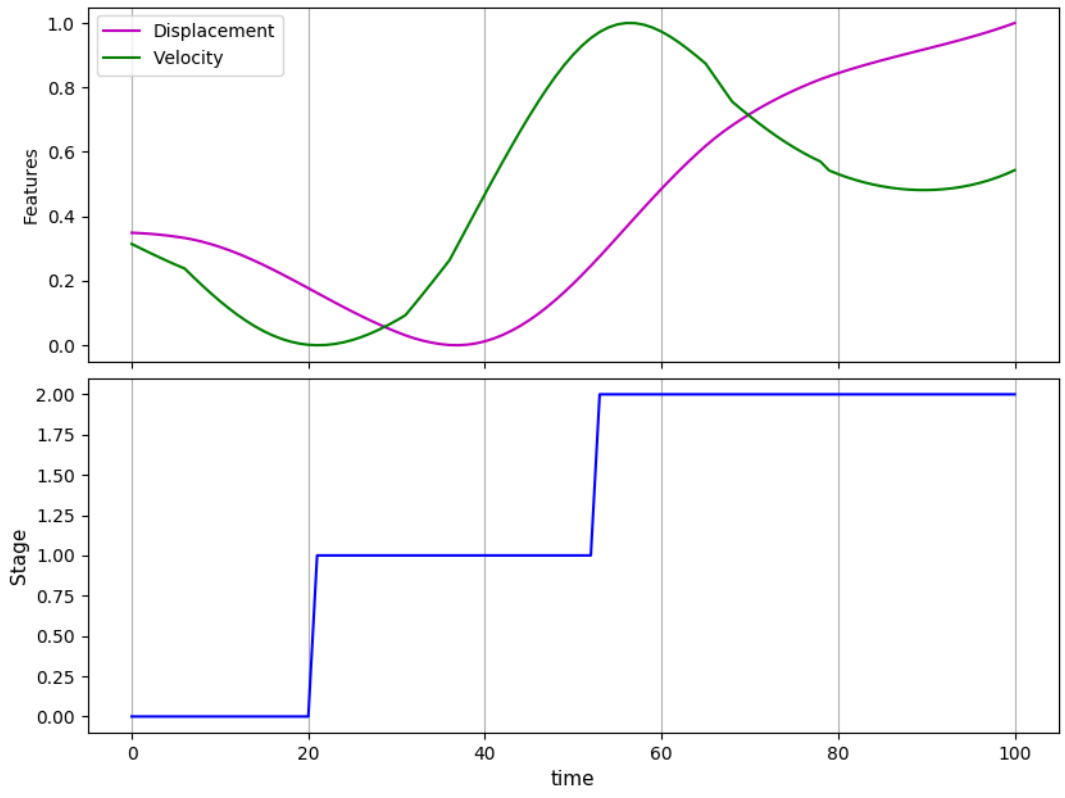

Figure 4.3. *Stages/Segments for a single successful demonstration Supervised clustering split when acceleration = 0, (time = 20 and time = 55)*

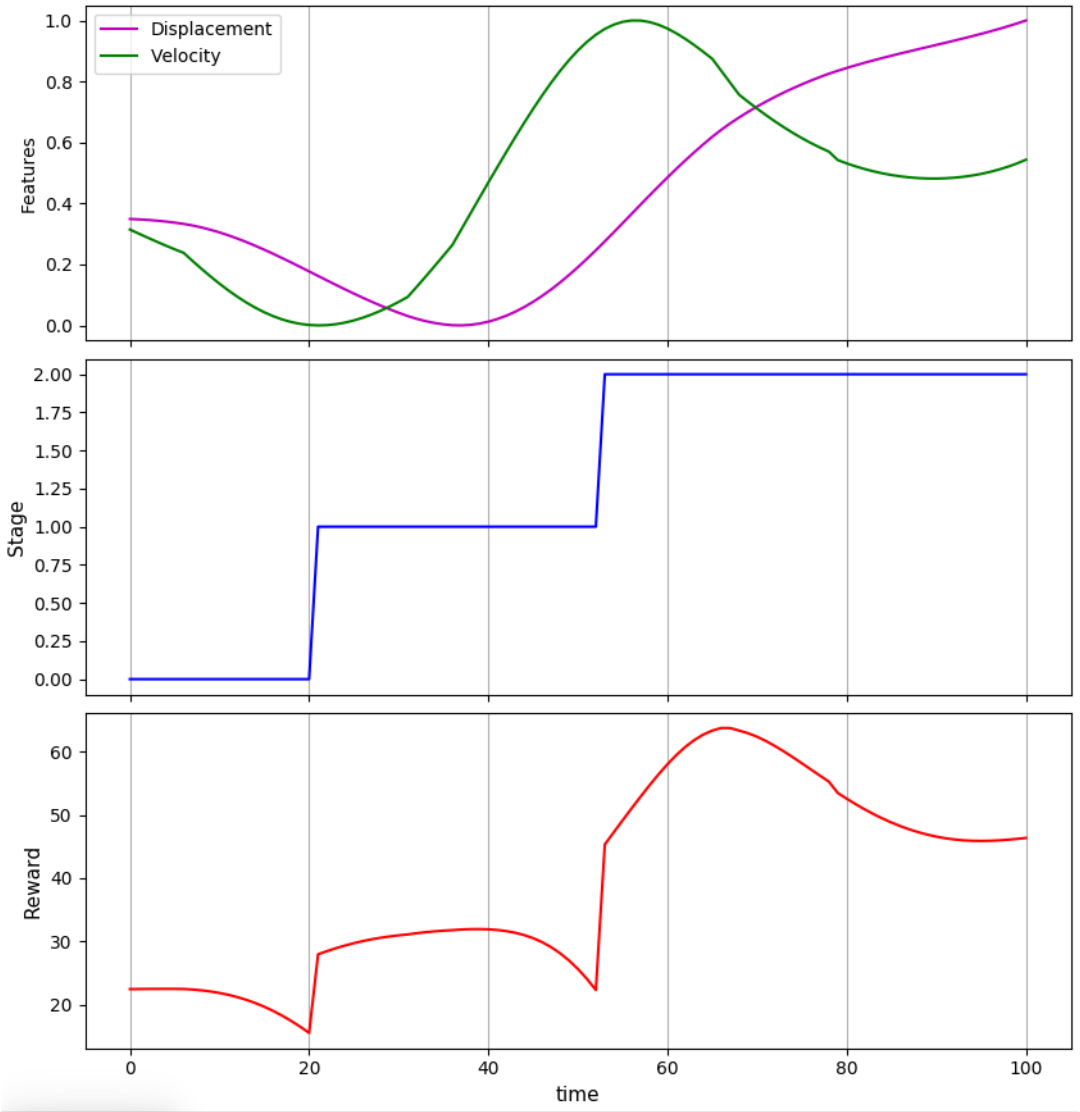

Figure 4.4. *Reward calculation using Equation 2.31*

for the reward function and a suitable classifier as well as a good R_{max} value (refer *Figure 4.5* for a bad example of R_{max}) was determined Q-Learning was carried out.

Table 4.1. *Stage Classification Accuracy Mountain Car*

Stage Classifier	**Accuracy %**
Decision Tree Classifier: max depth = 5	97.0
KNeighbours Classifier: neighbours = 5	96.0
Linear SVC: C = 0.025	23.7
Random Forest Classifier: max depth = 20, n estimators = 100	97.0
MLP Classifier: alpha = 1, max iterations = 100	94.0

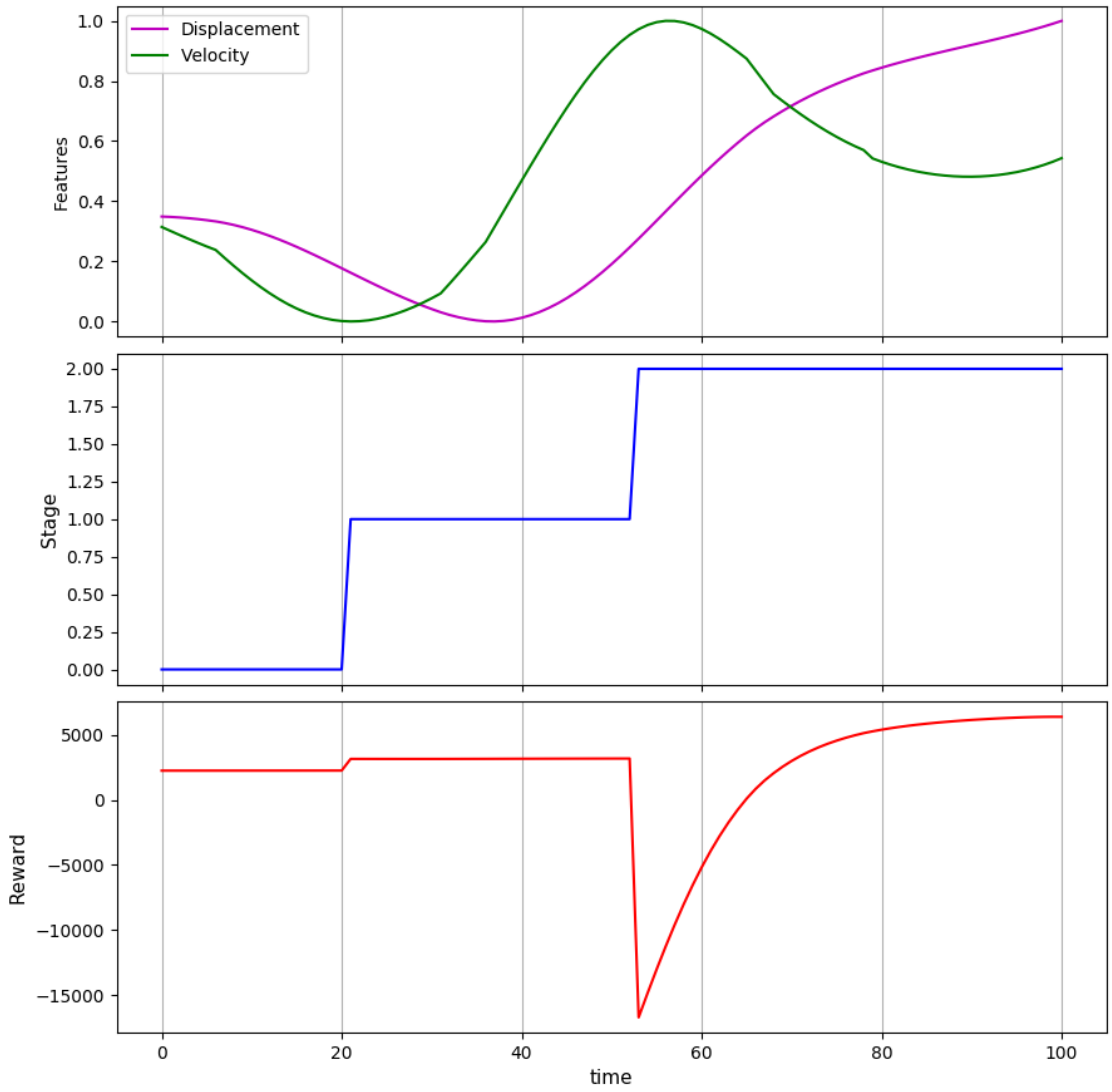

Figure 4.5. *Bad Reward Function for the mountain car problem ($R_{max} = 1600$), R_{max} is too low (see Section 3.4.4 for initializing R_{max})*

4.1.2 Q-Learning

Q-learning was carried out as shown in *Algorithm 2*. Q-learning was selected here simply because it is a simple and straightforward reinforcement learning algorithm. Since good results were obtained using q-learning other options were not explored. Using q-learning, the task was completed with 16000 episodes (see *Figure 4.7*) but for optimum results (refer *Figure 4.8*) the algorithm had to be run for about 32000 episodes.

4.2 The Machine

Q-learning on the mountain car confirmed the feasibility of unsupervised perceptual rewards for reward calculation. It is now possible to move on to implementation on the GIM Machine.

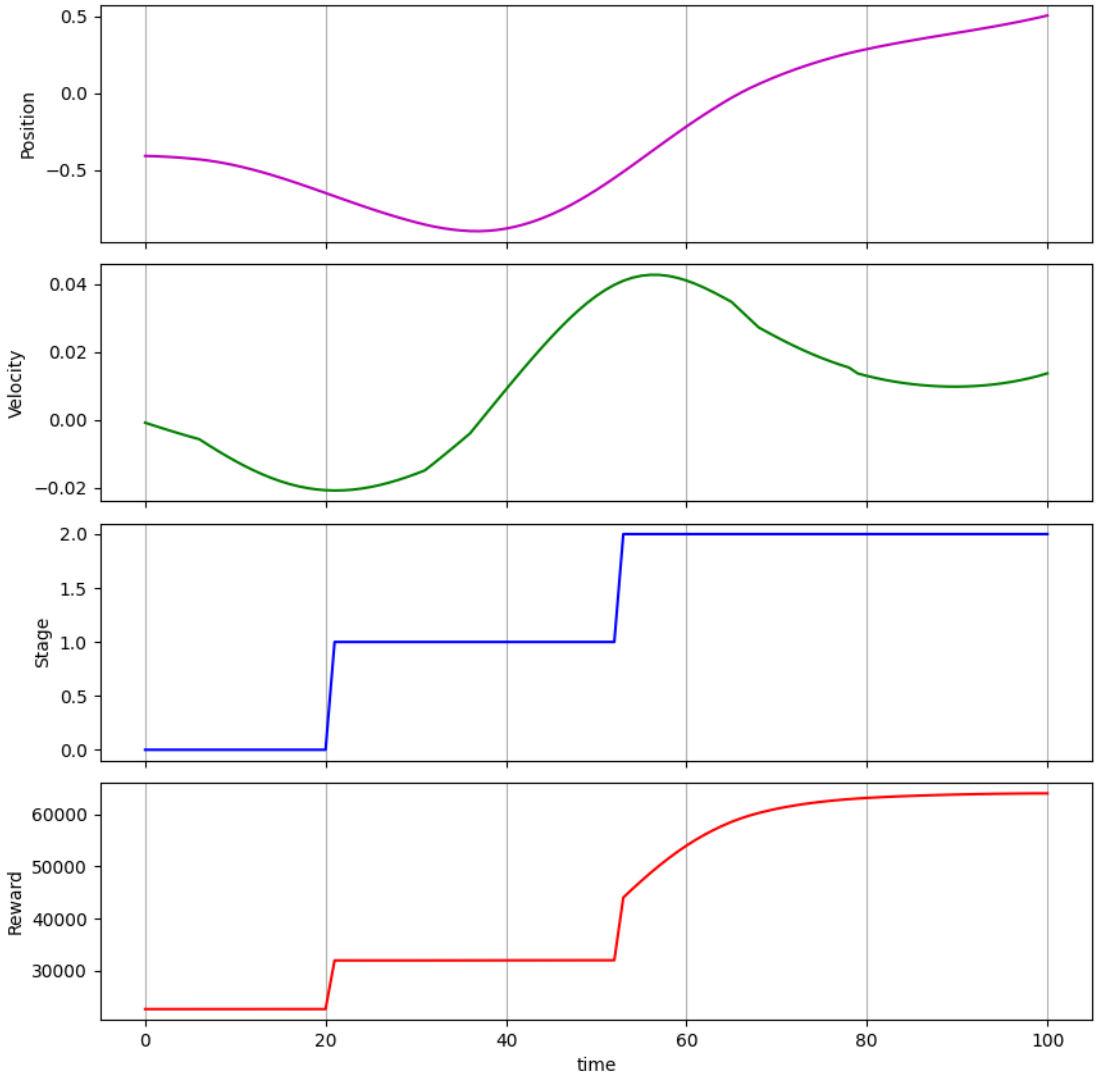

Figure 4.6. *Good Reward Function for the mountain car problem (Decision Tree Classifier and* $R_{max} = 16000$*)*

4.2.1 Feature Selection

In order to perform reasonable reward calculation, it is essential to carry out good feature selection. For the unsupervised perceptual rewards method, this means good stage classification. Feature selection consisted of a substantial part of testing. The GIM Machine's sensors measure a large number of parameters, including telescopic and transmission pressures, boom and bucket angles, the velocity of the machine, boom length and control signals such as throttle or gas. Various combinations of sensor data were considered during feature selection to obtain good stage classification.

For initial testing the same features (except visual features) were used for testing as done by *Yang et al* [2] (*Section 2.1.3*).

- Boom joint angle and bucket joint angle
- Hydraulic transmission drive pressure

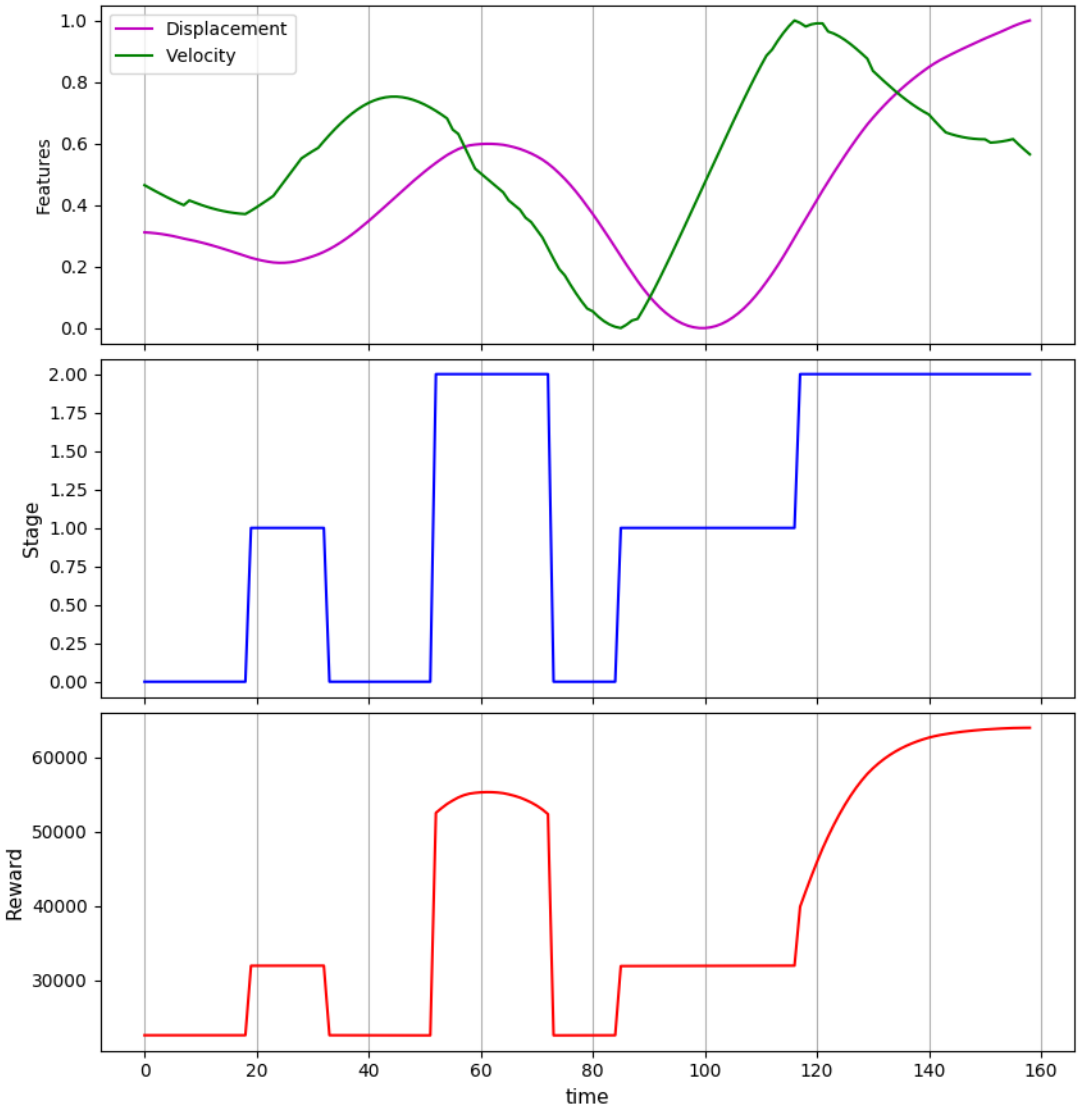

Figure 4.7. *Q-Learning for the mountain car problem ($R_{max} = 16000$) completed in 156 time steps (running 16000 episodes, epsilon = 0.9)*

- Throttle or gas command

It became clear based on these results that this configuration of features did not yield good stage classification (see *Figure* 4.9). Both the transmission pressures and the gas appeared unreliable as they are both of speed or acceleration type, unlike boom and bucket angles which are stationary states. So these two features were removed from features (refer *Figure 4.10*). Instead, distance to the pile was added to the features. Distance to the pile is a stationary state as it is an integral of speed, making it a good feature to use for the unsupervised perceptual reward method. The ZED camera's depth data is used to calculate the distance to the pile.

Stage classification is significantly better when using only boom angle, bucket angle and distance for reward calculation, but it may not yield satisfactory results in practice as the boom and bucket angle changes are a result of a movement in the telescope. The telescopic pressure is an early indication of these movements, and therefore including

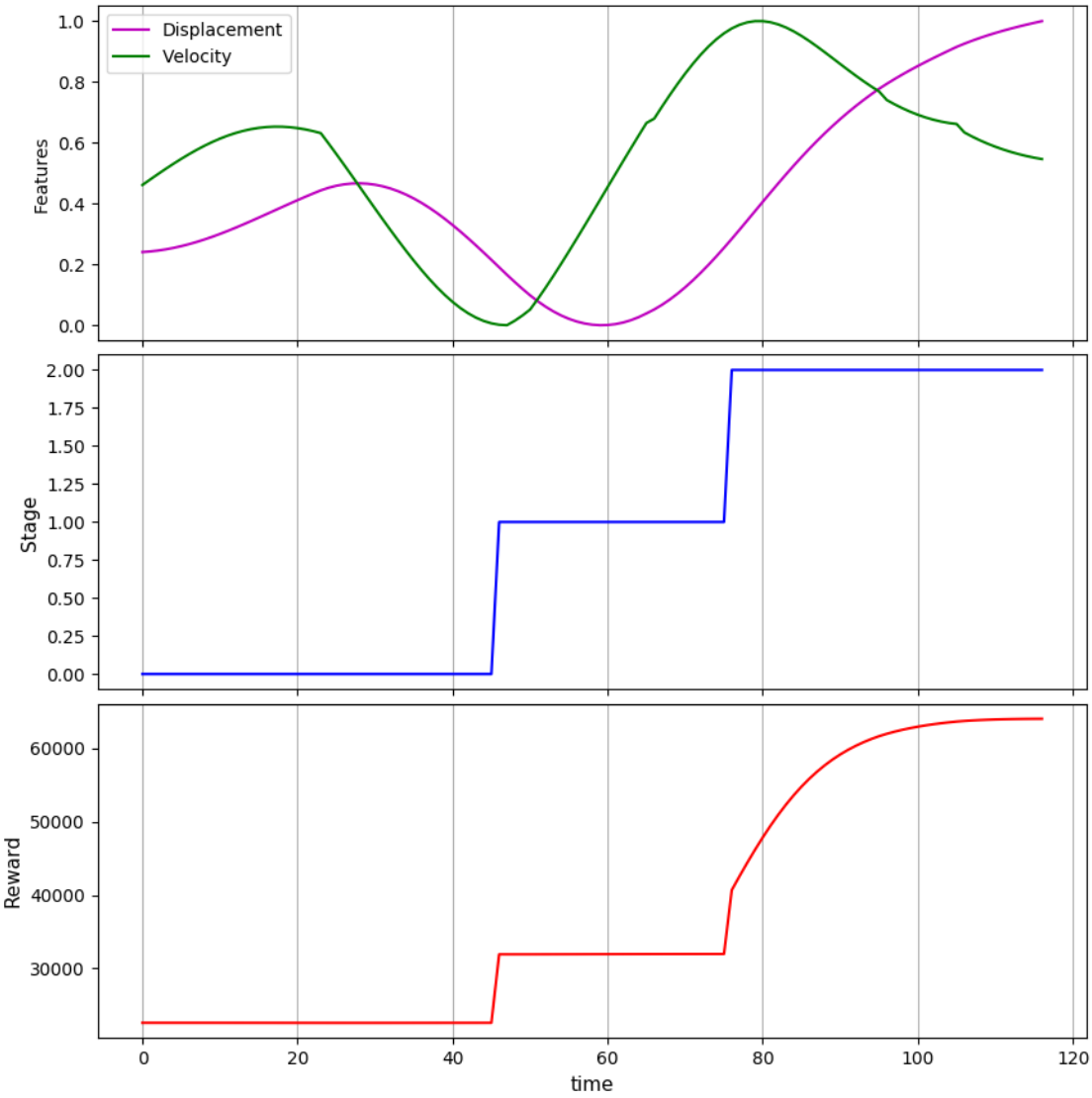

Figure 4.8. *Best results obtained using Q-Learning for the mountain car problem ($R_{max} = 16000$) completed in 117 time steps (running 32000 episodes, epsilon = 0.9)*

telescopic pressure in reward calculation would be more practical.

As can be seen in *Figure 4.11* telescopic pressure is not a useful feature for reward function using unsupervised perceptual rewards. There is a noticeable error in stage clustering and classification. Interestingly there is a decrease followed by an increase in the telescopic pressure (refer *Figure 4.11* at time = 7.5 s in winter and at time = 6 s in autumn) just before the increase in boom and bucket angles.

Similar to transmission pressure and gas tested before (refer *Figure 4.9*), the telescopic pressure is an event, unlike the boom angle, bucket angle and distance to the pile which are stationary states. While the angles are integrals of angular velocity, and the distance is an integral of velocity, the pressure is an early and accurate indication of stage changes. Therefore, memory is required to understand and use the pressure data correctly.

Work done (refer *Figure 4.12*) could then be a useful feature that represents the pressure

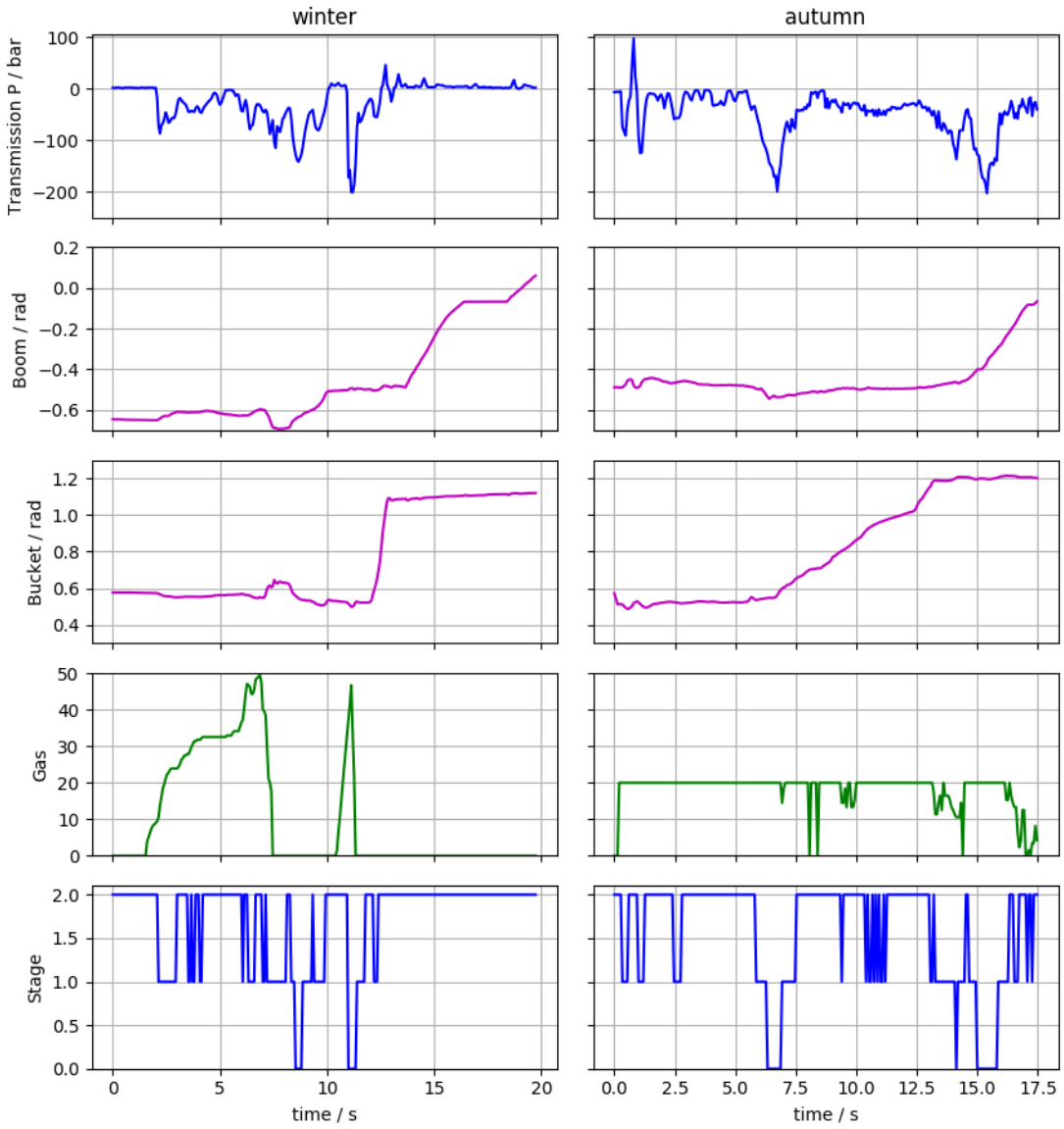

Figure 4.9. *Stage classification based on previous research features trained on mixed demonstrations*

data and holds its memory. There is an improvement in the segmentation when work done is added to the features (refer *Figure 4.12*). The winter test shows promising results with three clear stages. However, although the autumn data was a successful roll-out, it does not reach the third stage.

The significant difference in work done between winter and autumn can explain the unsuccessful stage classification of autumn data. The work done during the autumn roll-out is about half that observed in the winter demonstration. The higher work done during winter maybe because the ground is frozen, and the friction of the machine has increased during winter. Therefore, the work done had to be normalised to make winter and autumn data comparable.

Once the work done is normalized it is possible to obtain very good results (refer *Figure 4.13*). There are three clear stages seen in both the winter demonstration and in the

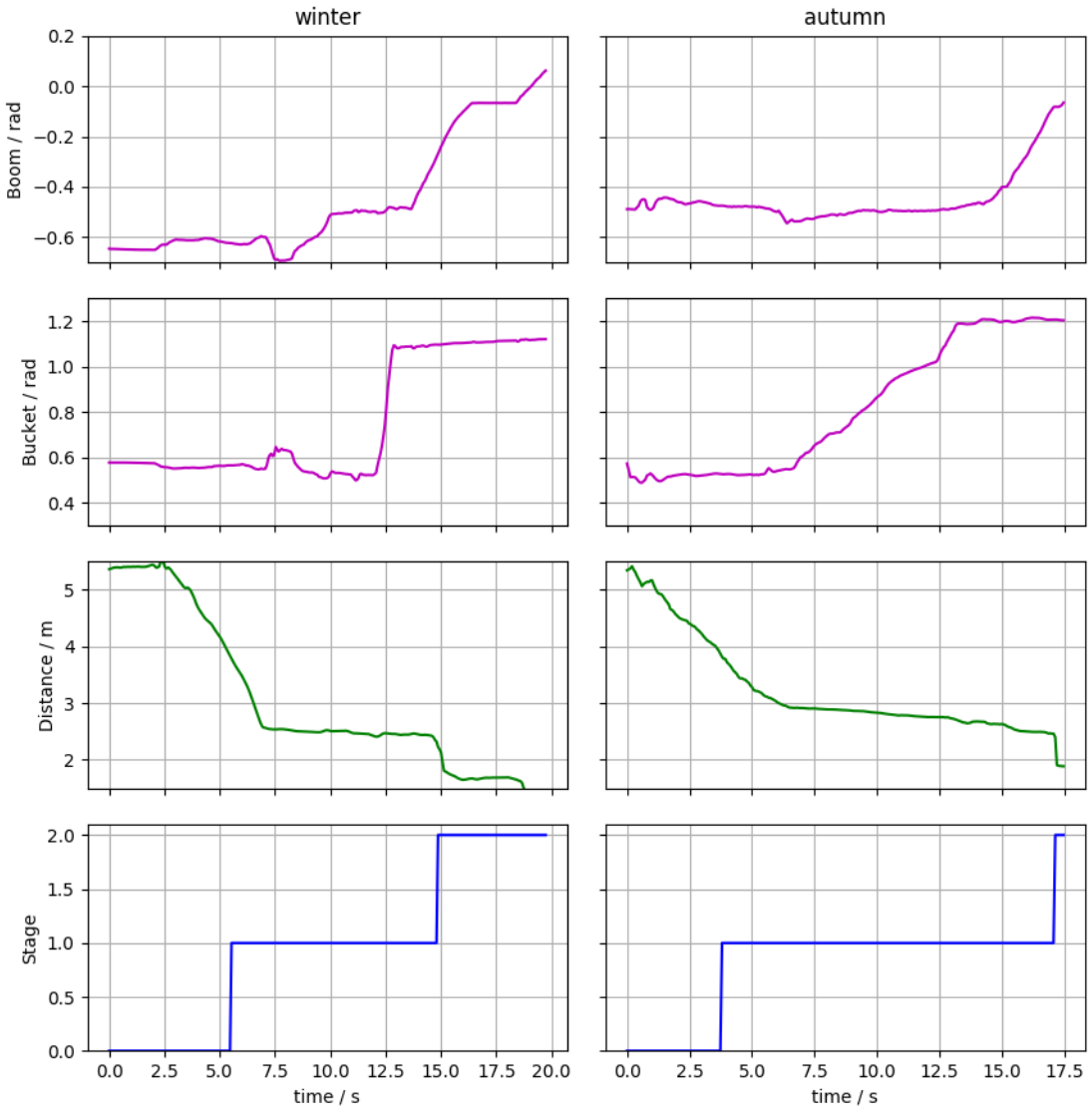

Figure 4.10. *Stage classification based on boom angles, bucket angles and distance to the pile trained on mixed demonstrations*

autumn roll-out. After good stage clustering and classification was obtained an R_{max} value was determined based on *Section 3.4.4*. Then the reward was calculated using *Equation 4* to obtain good increasing reward as seen in *Figure 4.13*. Good classification is a result of good feature selection.

Sermanet et al[17] also provide an equation to identify the most descriptive features for the unsupervised perceptual rewards method (see *Section* 2.4.4). z_i is the feature score for segment i. The higher feature score indicates that a feature is a more descriptive feature.

$$z_i = \alpha|\mu_i^+ - \mu_i^-| - (\sigma_i^+ + \sigma_i^-) \tag{4.1}$$

Table 4.2 shows the features arranged from the highest feature score to lowest feature for each stage, followed by the highest total feature score to lowest total feature score. Based

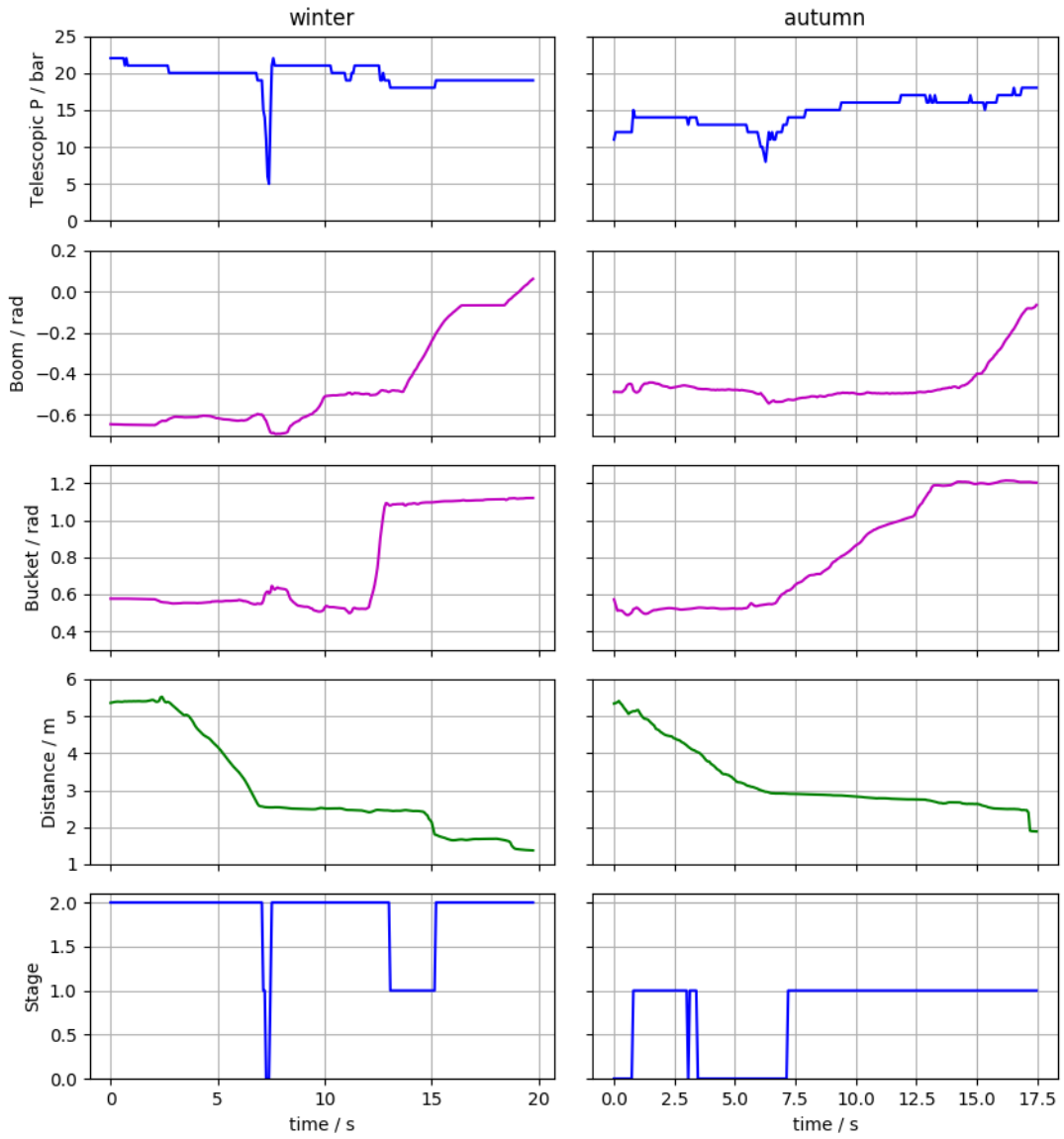

Figure 4.11. *Stage Classification based on telescopic pressure, boom and bucket angles and distance to the pile trained on mixed demonstrations*

Table 4.2. *Most Descriptive Features for each stage*

#	0	1	2	Total
1	Workdone	Workdone	Workdone	Workdone
2	Distance	Distance	Distance	Distance
3	Gas	Bucket	Transmission	Bucket
4	Boom	Telescope	Gas	Boom
5	Telescope	Gas	Bucket	Telescope
6	Bucket	Transmission	Boom	Transmission

on *Table 4.2* the work done and the distance to the pile are the most descriptive features. Proving that useful low-level sensor features have now been selected for testing.

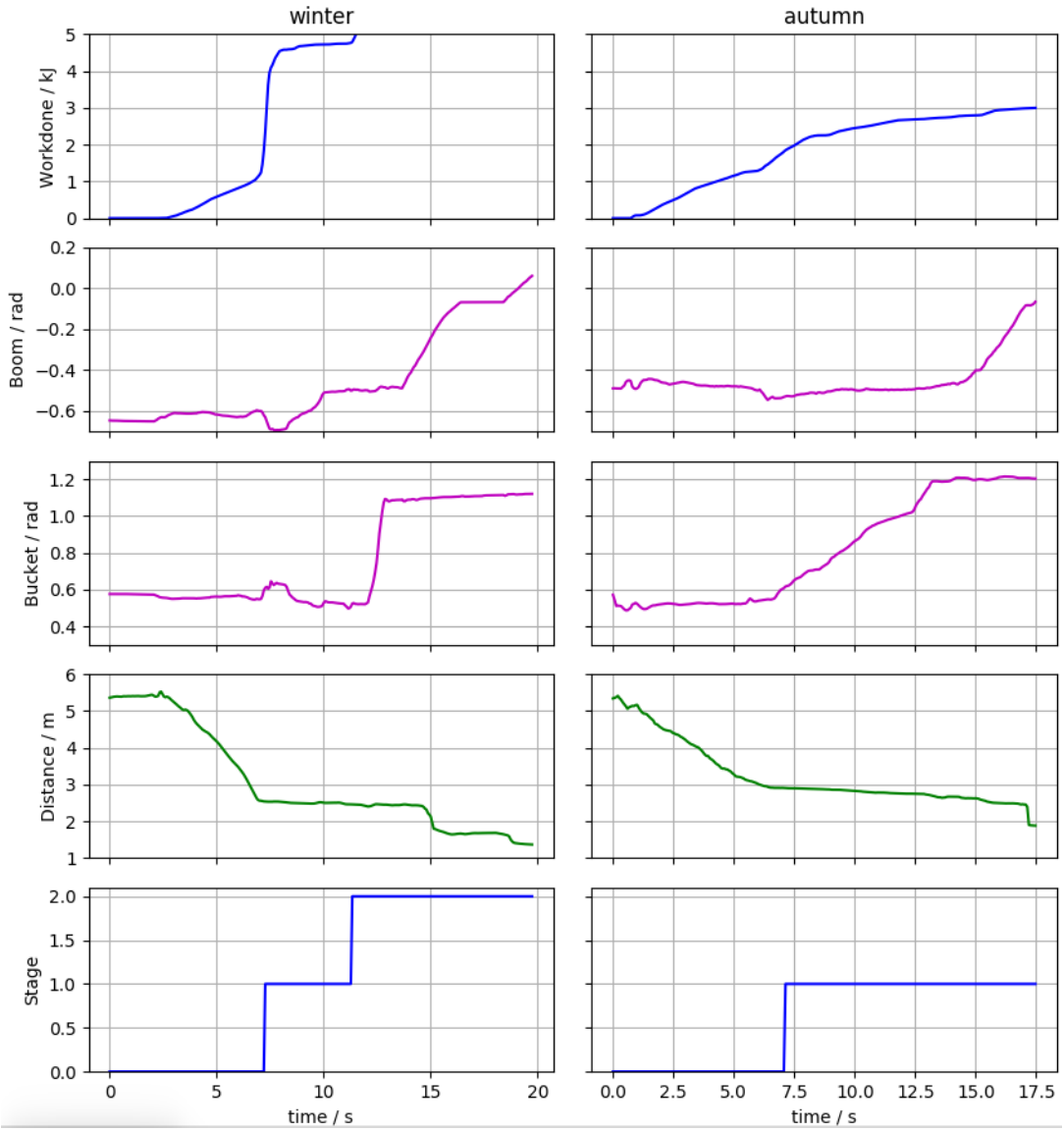

Figure 4.12. *Stage Classification based on workdone, boom and bucket angles and distance to the pile trained on mixed demonstrations*

4.2.2 Understanding the Data

Descriptive features are selected that result in a good increasing reward. It is, however, important to understand how the features vary as the task progresses. The graphs in *Figure 4.14* represent the division of stages or sub-tasks as described in detail below.

- Stage 1
  - Moving towards the pile
  - Distance to the pile decreases
  - Work done increases
- Stage 2
  - Hits the pile

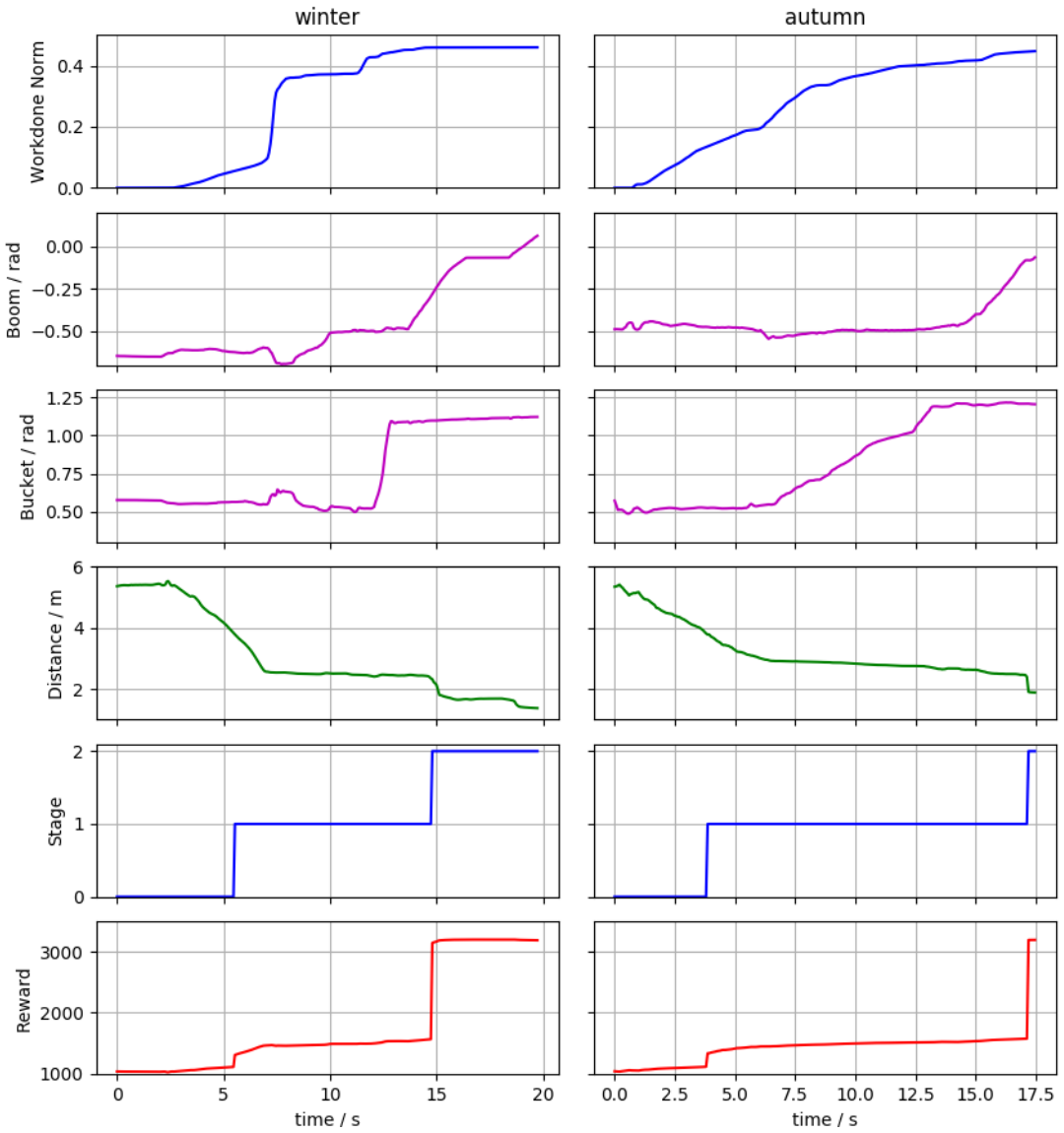

Figure 4.13. *Results based on workdone(normalized), boom and bucket angles and distance to the pile trained on mixed demonstrations:* $R_{max} = 600$

  - Distance to the pile plateaus
  - Fill the Bucket
  - Bucket Angle increases
  - Lift the pile
  - Boom Angle increases
  - Distance to the pile may decrease slightly
- Stage 3
  - Boom Angle starts to plateau
  - Task Completed

However, after feature selection, the actual values of the graphs matter less, and the

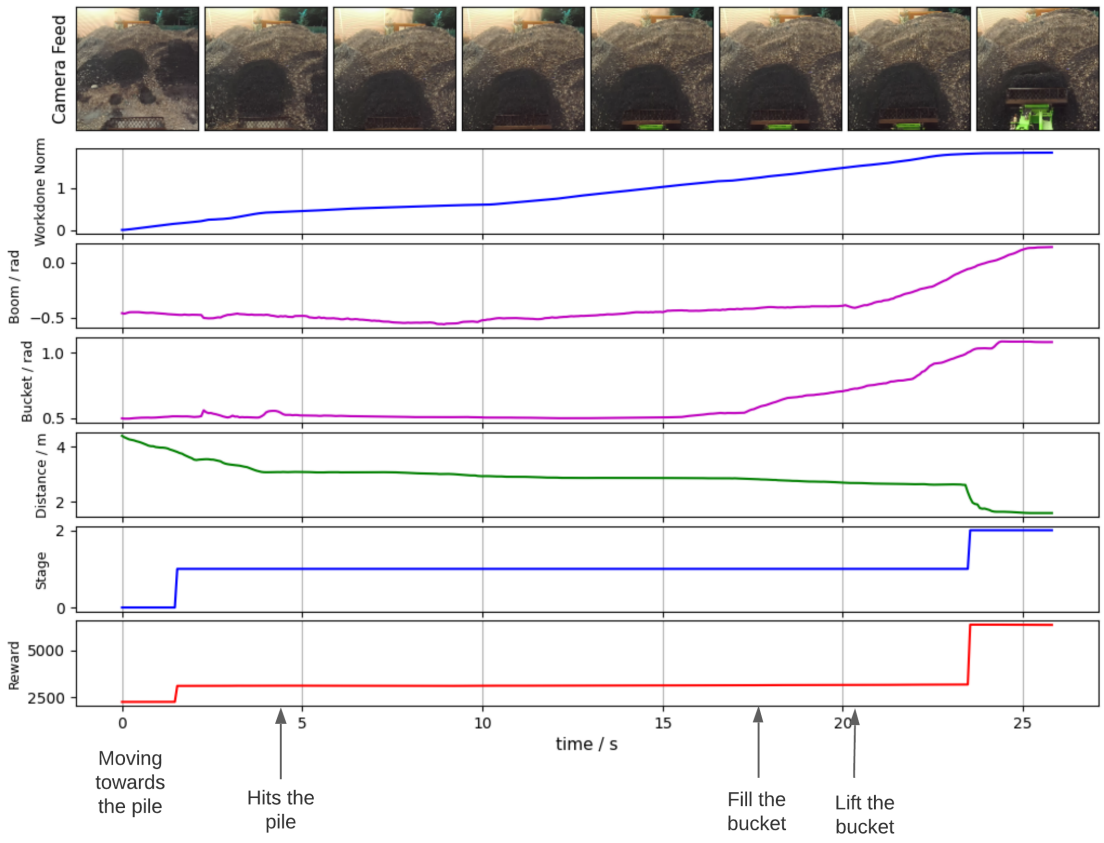

Figure 4.14. *Understanding the graphs*

shapes of the graphs matter more (when the feature values increase or decrease). Hence the rest of the thesis will follow the simplified representation shown in *Figure 4.15*

4.2.3 Image Features

Once the low-level features were selected, image features were also tested. Initially, image features were only used to determine task completion; later, they were also used for reward calculation.

The pre-trained three-dimensional convolutional neural network based on *Yang et al.'s* research [2] was used to extract image features. The input is a sequence of five colour images. It is a sliding window of size five, sliding through the video. Eight features were extracted in this way at a time. These features are then used to fit and test a task completion predictor.

4.2.4 Task Completion

Tests were carried out to determine the task completion accuracy on various classifiers to determine the best classifier. When training was carried out on 80 % of the mixed data followed by testing on 20 % of the mixed data (refer Figure 4.17 and Figure 4.16 for good

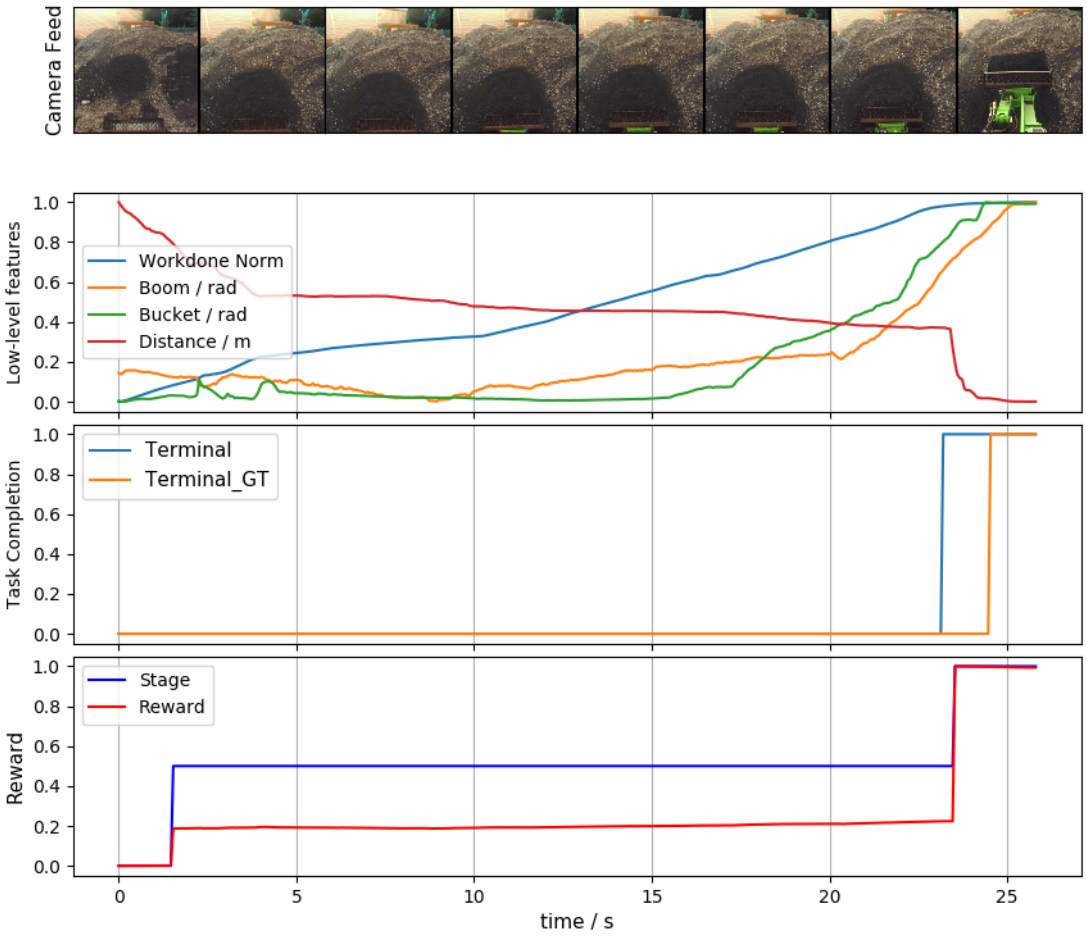

Figure 4.15. *Simplified representation*

results, data has been scaled to 0-1). The task completion classification accuracy varied between 92-93 % (refer 4.3) with a random forest classifier having the highest accuracy at 92.7 %.

Table 4.3. *Task Completion Accuracy: 80 % training and 20 % testing (refer Figure 4.16) and Figure 4.17*

Terminal State Classification	**Accuracy %**
KNeighbours Classifier: 5 neighbours	92.2
SVC: linear, C=0.025	92.1
Decision Tree Classifier: max depth = 5	92.6
Random Forest Classifier: max depth = 20, n estimators = 100, max features = 4	92.7
MLP Classifier	92.1
AdaBoost Classifier	92.4

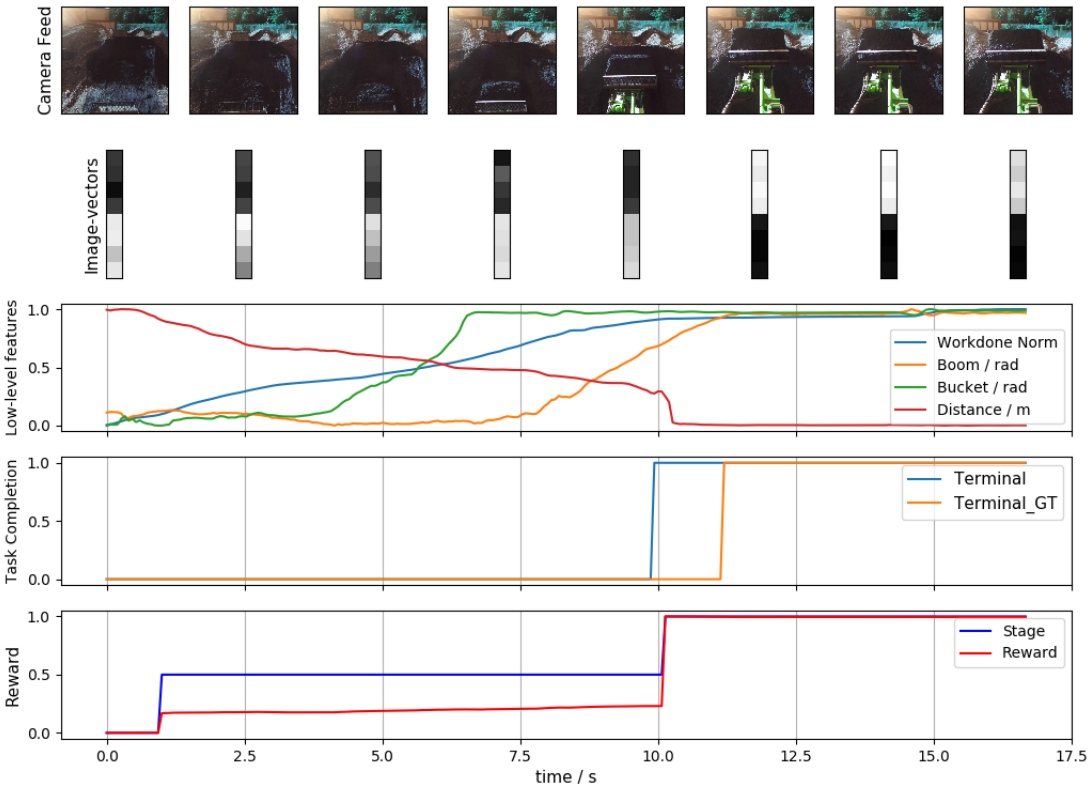

Figure 4.16. *Testing Autumn Demonstrations /w sensors Trained on Mixed Demonstration*

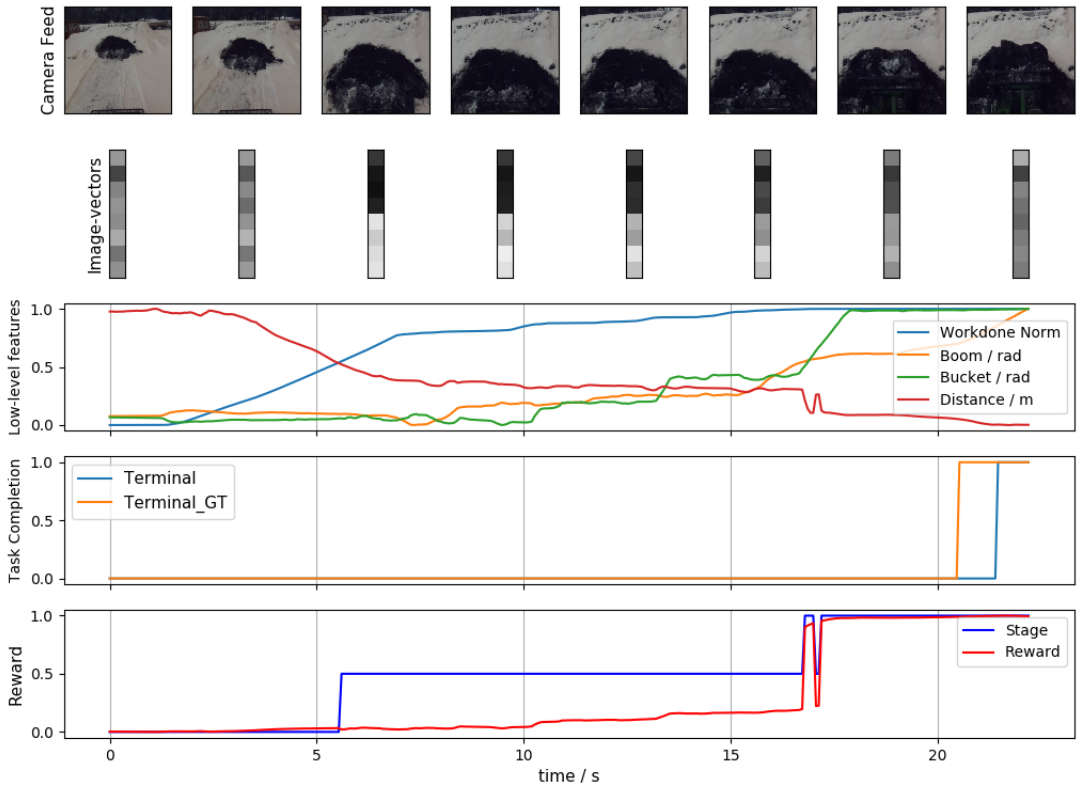

Figure 4.17. *Testing Winter Demonstrations /w sensors Trained on Mixed Demonstration*

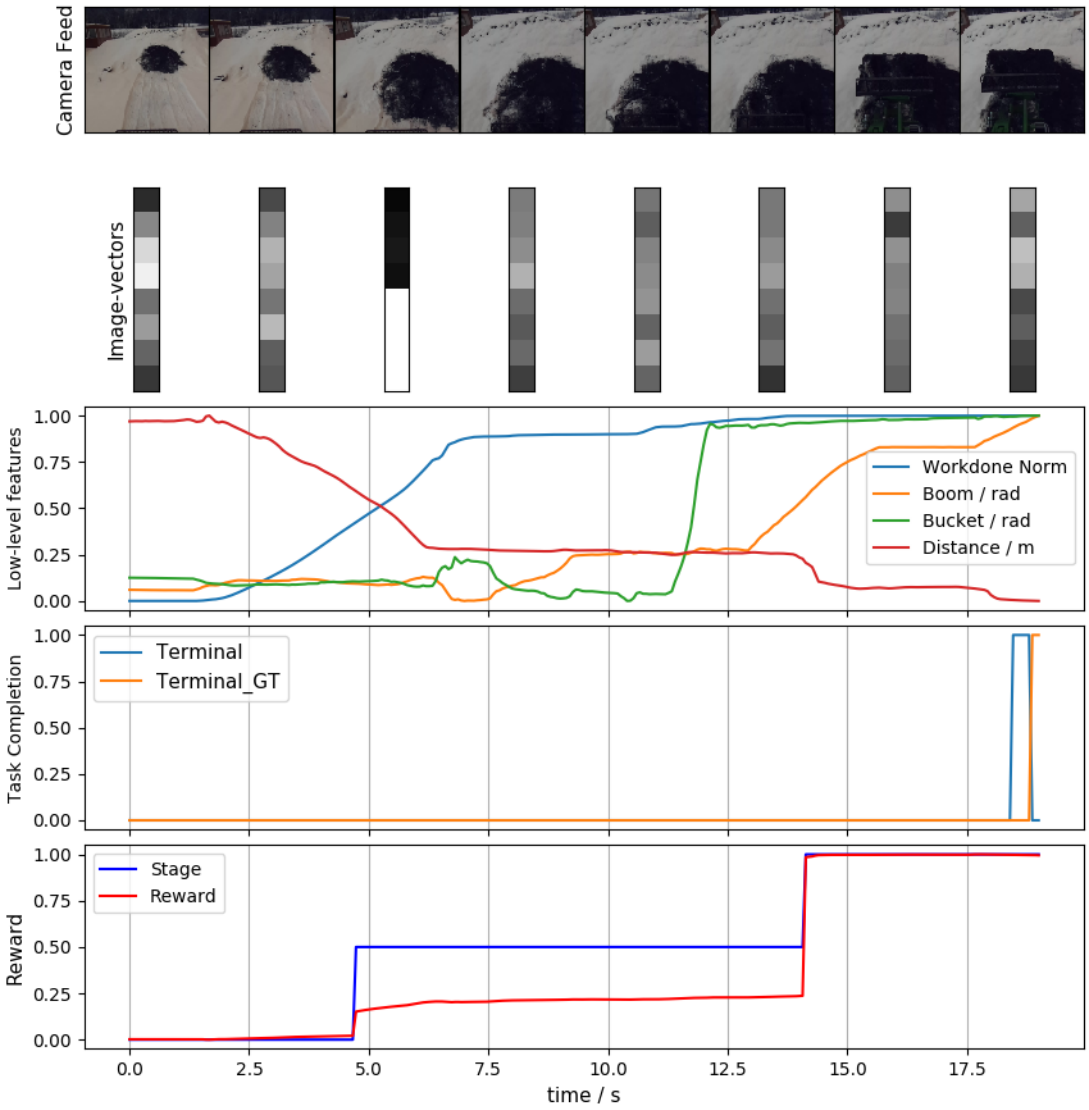

Figure 4.18. *Testing Winter Demonstration /w sensors+visual on mixed demonstrations*

4.2.5 With Visual Features for Reward Calculation

Following tests for Task Classification using image features, visual features were added for reward calculation. The R_{max} value had to be increased from 600 to 1600. Once a good R_{max} was obtained it was possible to obtain good classification as clear from *Figures 4.18 and 4.19*, but there was no visible improvement in the reard function.

4.2.6 Generalization

One of the main goals of this research is to determine if the method generalises well to changes in weather and lighting. In *Section 4.2.1* one of the initial problems with using autumn and winter data was encountered. Work done during winter was near twice the work done during autumn, and this meant that the work done needed to be normalised (see *Figure 4.13*) before it was used for unsupervised clustering and then classification.

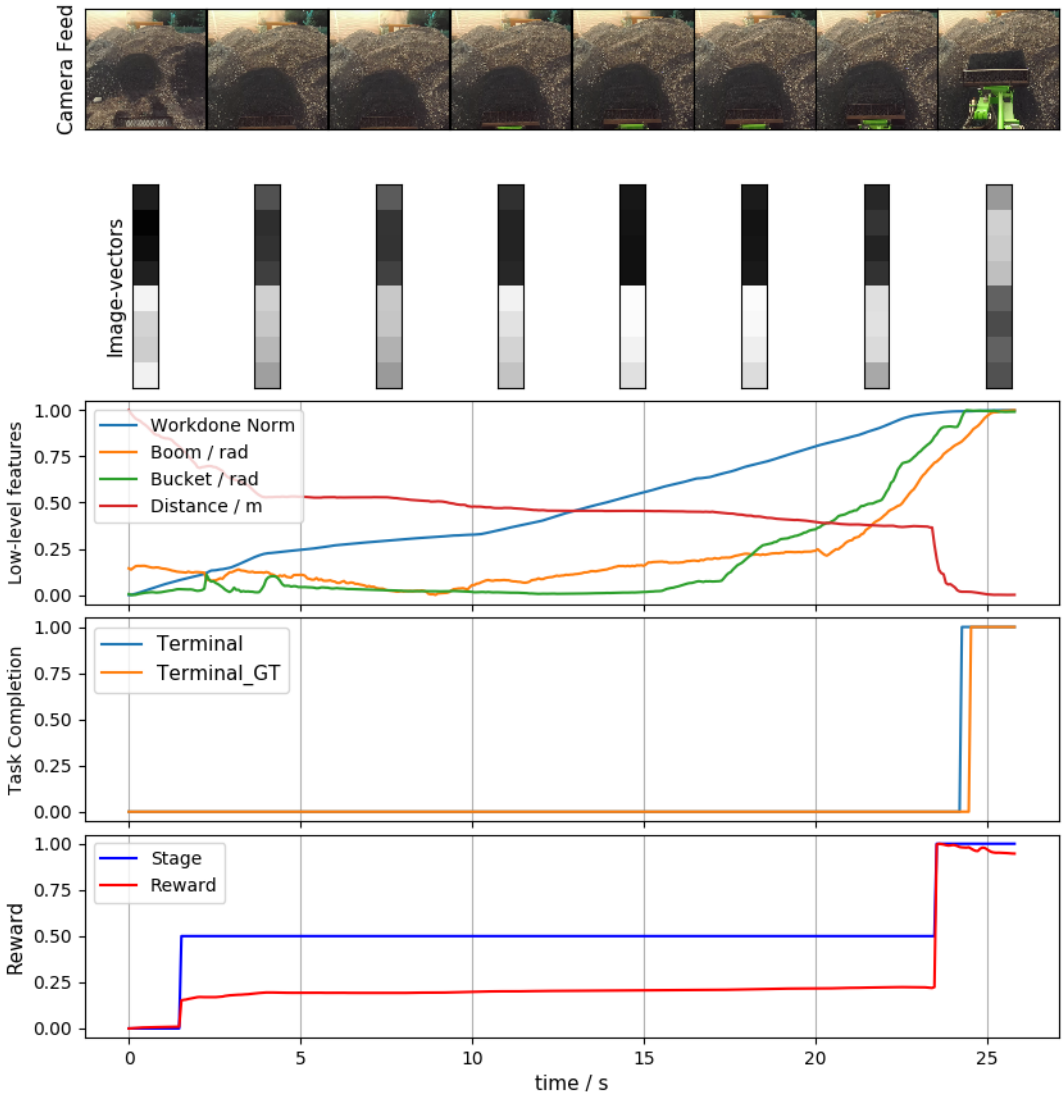

Figure 4.19. *Testing Autumn Demonstration /w sensors+visual trained on mixed demonstrations*

How well does the task completion predictor generalise to changes in weather and lighting? It is important to note that *Yang et al.*'s [2] model was trained and tested only on summer data. The tests to follow will also determine how well the pre-trained model generalises to autumn and winter data. The task completion accuracy results are very interesting as depicted in the *Tables 4.4*. When the classifier was trained using autumn data and tested on winter data, the classification accuracy increased to 94-96 %, with KNN classifier having the highest classification accuracy of 95.4 %. This high classification accuracy obtained when the task completion predictor was trained on autumn data could mean that the feature extractor extracts more descriptive features when trained on autumn data. However, when the classifier was trained using winter data and tested on autumn data (refer *Table 4.4*), the accuracy dipped significantly. The accuracy varied between 77-89 % with SVC showing the most accuracy at just 88.2 %. The classifier was not able to generalise well using the winter data for training.

Table 4.4. *Task Completion Accuracy*

Terminal State Classification	**Mixed** %	**Testing Winter %**	**Testing Autumn %**
refer *Figure/s*	4.18 and 4.19	4.20	4.22 and 4.21
KNeighbours Classifier: 5 neighbours	92.2	95.4	77.9
SVC: linear, C=0.025	92.1	94.3	88.2
Decision Tree Classifier: max depth = 5	92.6	94.7	79.3
Random Forest Classifier: max depth = 20, n estimators = 100, max features = 4	92.7	94.7	77.8
MLP Classifier	92.1	94.7	77.5
AdaBoost Classifier	92.4	94.8	77.5

Stage classification worked quite well, once work done was normalised, when using only low-level features. However, is the method able to still generalise well once visual features are added? Although the visual features weren't able to generalize well for task completion during autumn testing (refer *Table 4.4* and *Figure 4.22*) they were able to generalize well during stage classification producing good results both for testing during winter and autumn (refer *Figure 4.20 and Figure 4.21 respectively*). Although visual features could be used for reward calculation, they do not result in any improvement in the reward function. Therefore it is possible to obtain a good reward function using only the low-level features.

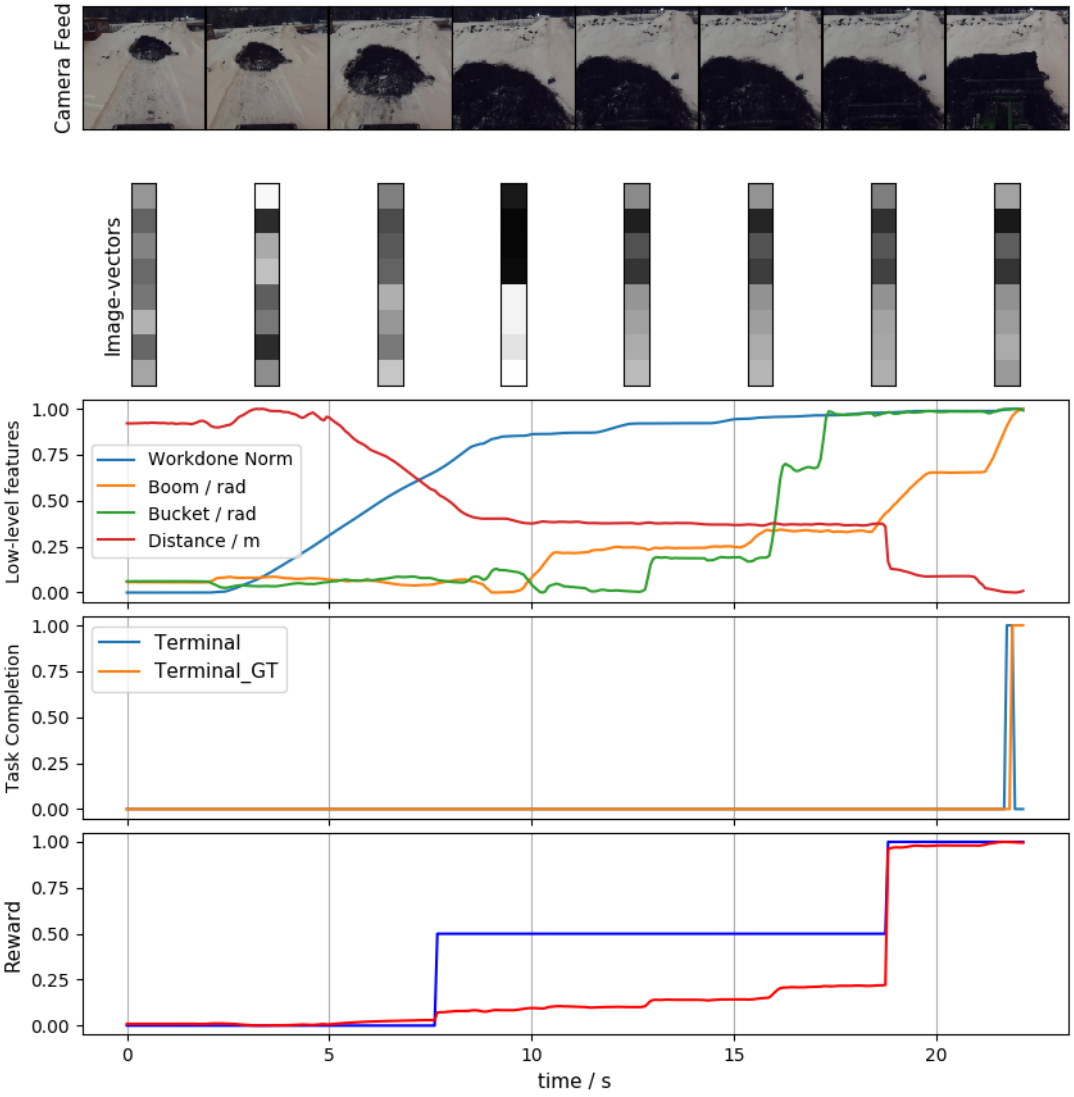

Figure 4.20. *Testing in Winter and Training in Autumn w/ Sensors: Successful Task Completion Prediction*

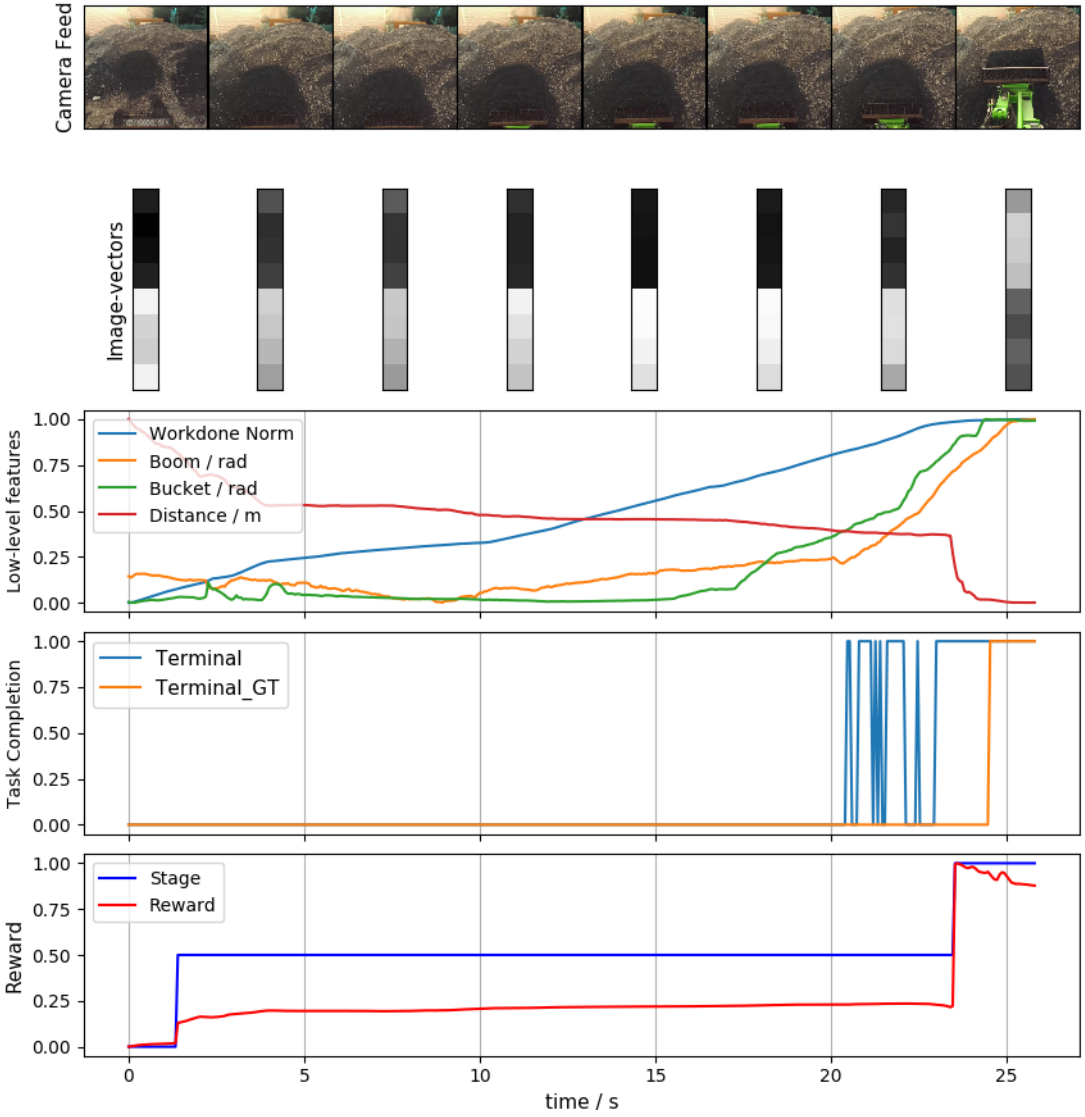

Figure 4.21. *Testing in Autumn and Training in Winter w/ Sensors+Visual: Early Task Completion Prediction*

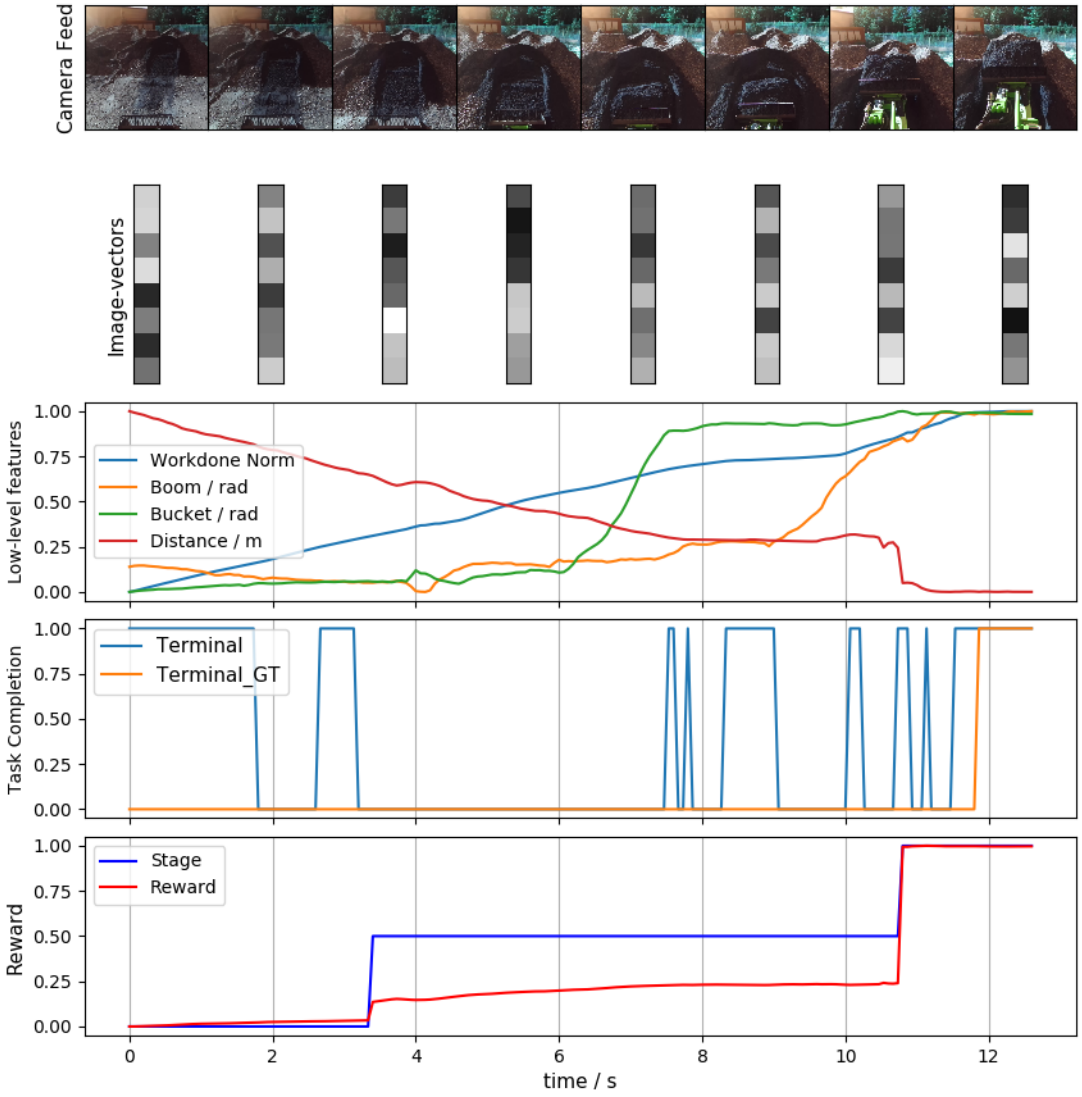

Figure 4.22. *Testing in Autumn and Training in Winter w/ Sensors+Visual: Failure Task Completion Prediction*

5 CONCLUSIONS

The main goal of this thesis was to find a reward function for autonomous earthmoving. The research began with five questions(refer *Section 1.2*) and this thesis attempts to answer these questions and as a result, achieve its primary goal.

The first of the research questions asks: **What sensor data is appropriate for the task?** There was a variety of sensor data available at our disposal including transmission and telescopic pressures, boom and bucket angles, boom length, the velocity of the GIM Machine and camera images. Various configurations of data (refer *Section 4.2.1*) were tested to select work done, boom angle, bucket angle and distance to the pile as the low-level features. Once low-level features were determined a means to determine image features was required. Fortunately, *Yang et al.* [2], who was a big inspiration for this research, already extracted good visual features using a 3D convolutional neural network. This same network was used in this research to extract eight image features (per 5 frames, sliding window). The image features extracted were initially used to confirm task completion (refer *Section 3.5*) but later used for reward calculation (refer *Section 3.6*) as well.

The data available during training consists of low-level features and extracted image features. However, a model-free inverse reinforcement learning method is preferred as there is no knowledge of the transition probabilities. The second question can now be answered: **What properties should the reward function have?**. The reward function calculation should not require system dynamics (no knowledge of transition probabilities). The reward function must increase as the task progresses. However, the linearity of the reward function with respect to the features does not matter. Other factors to consider when determining whether a method will be the right candidate for reward calculations, such as sample efficiency and the preference of an offline algorithm, are discussed in *Section 2.2.1*.

The available sensor data, system dynamics, sample efficiency and ease of implementation will be used to answer the fourth research question: **What method is appropriate to compute the reward function for autonomous earth moving?** After studying various inverse reinforcement learning methods (refer *Chapter 2*) unsupervised perceptual rewards [17] became the method of choice. This method is a sample efficient method that could not only be applied to a machine with unknown system dynamics but was also easy to implement on a machine without good simulations of the machine's environment and its interactions.

Once unsupervised perceptual rewards was selected the next research question could be answered: **Can the method be used to solve a more straightforward model problem?** Before applying the algorithm to the GIM Machine data, it was applied to a straightforward problem, the OpenAI gym mountain car problem (refer *Section* 4.1). The reward function was modified slightly (see *Figure 4.4* for reward based on *Sermanet et al [17]* and *Equation 3.1* for the modified reward function) such that the distance to the next stage was used to calculate reward. Once the reward function was tested on successful demonstrations q-learning was used to confirm the effectiveness of the calculated reward for reinforcement learning on the mountain car problem.

Since the tests on mountain car confirmed the effectiveness of unsupervised perceptual rewards as a good method for reward calculation it is now possible to move on to the fifth and final research question: **Can the method solve the autonomous earth moving problem and does it generalise well to differences in weather conditions?** It was possible to use *Sermanet et al's* [17] method unsupervised perceptual reward to obtain a good reward function for autonomous earthmoving (see results *Section 4.2*). The method was able to generalise to an extent when using variations of winter and autumn data with regard to reward calculation and stage classification. The main obstacle with regards to generalisation was the workdone (see *Figure 3.5*). The workdone during winter was about twice as much as the workdone during autumn. This meant that the workdone had to be normalized in order to be comparable and to produce a good reward function(see *Figure 4.13*). Visual features were also tested for reward calculation but didn't result in any improvement. The task classifier produced some interesting results. The task classification accuracy ranged from 92-96 % when either, training and testing was both carried out on mixed demonstrations (see *Table 4.3*), or winter data was tested on a classifier trained on autumn data (see *Table 4.4*). However, testing autumn data on a classifier trained on winter data (*Table 4.4*) did not produce good task classification results. These results are also an indication of how well *Yang et al*'s[2] pretrained model for image feature extraction trained on summer data could generalise to autumn and winter data.

In conclusion unsupervised perceptual rewards appears to be a suitable method for reward calculation for autonomous earthmoving and it is possible to obtain a good reward function using only the low-level features. For future work we will apply the reward function obtained in this research to carry out reinforcement learning on the real machine.

REFERENCES

[1] Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P. and Peters, J. An Algorithmic Perspective on Imitation Learning. *Foundations and Trends in Robotics* 7.1-2 (2018), 1–179. ISSN: 1935-8261. DOI: 10.1561/2300000053. URL: http://dx.doi.org/10.1561/2300000053.

[2] Yang, W., Strokina, N., Serbenyuk, N., Ghabcheloo, R. and Kämäräinen, J. K. Learning a Pile Loading Controller from Demonstrations. *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[3] Codevilla, F., Santana, E., López, A. M. and Gaidon, A. *Exploring the Limitations of Behavior Cloning for Autonomous Driving*. 2019. arXiv: 1904.08980 [cs.CV].

[4] Finn, C., Levine, S. and Abbeel, P. Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, 49–58.

[5] Boularias, A., Kober, J. and Peters, J. Relative Entropy Inverse Reinforcement Learning. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, 182–189. URL: http://proceedings.mlr.press/v15/boularias11a.html.

[6] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A. and Shah, A. *Learning to Drive in a Day*. 2018. arXiv: 1807.00412 [cs.LG].

[7] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I. and Thrun, S. Junior: The Stanford Entry in the Urban Challenge. *Journal of Field Robotics* 25 (Sept. 2008), 569–597. DOI: 10.1002/rob.20258.

[8] Chang, H. S. *Simulation-Based Algorithms for Markov Decision Processes*. eng. 2nd ed. Communications and Control Engineering. London: Springer London. ISBN: 1-4471-5022-8.

[9] Pomerleau, D. A. ALVINN: An Autonomous Land Vehicle in a Neural Network. *Advances in Neural Information Processing Systems 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, 305–313. ISBN: 1558600159.

[10] Viswanath, P., Nagori, S., Mody, M., Mathew, M. and Swami, P. End to End Learning based Self-Driving using JacintoNet. *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. Sept. 2018, 1–4. DOI: 10.1109/ICCE-Berlin.2018.8576190.

[11] Sutton, R. S. and Barto, A. G. *Reinforcement Learning:An Introduction*. Cambridge, Massachusetts: The MIT Press, 2015.

[12] Ratliff, N. D., Bagnell, J. A. and Zinkevich, M. A. Maximum Margin Planning. *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, 729–736. ISBN: 1595933832. DOI: 10.1145/1143844.1143936. URL: https://doi.org/10.1145/1143844.1143936.

[13] Ziebart, B. D., Maas, A., Bagnell, J. A. and Dey, A. K. Maximum Entropy Inverse Reinforcement Learning. *Proc. AAAI*. 2008, 1433–1438.

[14] Ho, J. and Ermon, S. Generative Adversarial Imitation Learning. *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, 4572–4580. ISBN: 9781510838819.

[15] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative Adversarial Nets. *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, 2672–2680.

[16] Paul, S., Baar, J. van and Roy-Chowdhury, A. K. *Learning from Trajectories via Subgoal Discovery*. 2019. arXiv: 1911.07224 [cs.LG].

[17] Sermanet, P., Xu, K. and Levine, S. Unsupervised Perceptual Rewards for Imitation Learning. *CoRR* abs/1612.06699 (2016). arXiv: 1612.06699. URL: http://arxiv.org/abs/1612.06699.

[18] *MountainCar-v0*. https://gym.openai.com/envs/MountainCar-v0/. Accessed: 28-09-2020.